3-21-2013

A Multi Agent System for Flow-Based Intrusion Detection

David A . Ryan

A MULTI AGENT SYSTEM FOR FLOW-BASED INTRUSION DETECTION

THESIS

David A. Ryan, Second Lieutenant, USAF

AFIT-ENG-13-M-43

DEPARTMENT OF THE AIR FORCE
AIR UNIVERSITY

AIR FORCE INSTITUTE OF TECHNOLOGY

Wright-Patterson Air Force Base, Ohio

AFIT-ENG-13-M-43

A MULTI AGENT SYSTEM FOR FLOW-BASED INTRUSION DETECTION

THESIS

Presented to the Faculty

Department of Electrical and Computer Engineering

Graduate School of Engineering and Management

Air Force Institute of Technology

Air University

Air Education and Training Command

in Partial Fulfillment of the Requirements for the

Degree of Master of Science in Computer Engineering

David A. Ryan, B.S.C.E.

Second Lieutenant, USAF

March 2013

AFIT-ENG-13-M-43

A MULTI AGENT SYSTEM FOR FLOW-BASED INTRUSION DETECTION

David A. Ryan, B.S.C.E.
Second Lieutenant, USAF

Approved:

Dr. Gary B. Lamont, PhD (Chairman)	5 MAR 2013 Date
Dr. Gilbert L. Peterson, PhD (Member)	8 MAR 2013 Date
Lt Colonel Jeffrey Clark, PhD (Member)	8 Mar 2013 Date

AFIT-ENG-13-M-43

Abstract

The detection and elimination of threats to cyber security is essential for system functionality, protection of valuable information, and preventing costly destruction of assets. This thesis presents a Mobile Multi-Agent Flow-Based IDS called MFIREv3 that provides network *anomaly detection* of intrusions and automated defense.

This version of the MFIRE system includes the development and testing of a Multi-Objective Evolutionary Algorithm (MOEA) for feature selection that provides agents with the "optimal" set of features for classifying the state of the network. Feature selection provides separable data points for the selected attacks: Worm, Distributed Denial of Service, Man-in-the-Middle, Scan, and Trojan.

This investigation develops three techniques of self-organization for multiple distributed agents in an intrusion detection system: Reputation, Stochastic, and Maximum Cover. These three movement models are tested for effectiveness in locating good agent vantage points within the network to classify the state of the network.

MFIREv3 also introduces the design of defensive measures to limit the effects of network attacks. Defensive measures included in this research are rate-limiting and elimination of infected nodes.

The results of this research provide an optimistic outlook for flow-based multi-agent systems for cyber security. The impact of this research illustrates how feature selection in cooperation with movement models for multi agent systems provides excellent attack detection and classification.

To Rosie.

Acknowledgments

I express my deepest gratitude to all those whom continue to support me and sacrificed time and effort to help me through a difficult time.

Professor Lamont provided the patience, encouragement, and knowledge making this effort possible. He has dedicated extraordinary time in my education and success. I am indebted to him for my growth as a student, officer, and engineer. My gratitude goes out as well to all the professors in the Computer Engineering department for their advice and teachings.

I would not have contemplated this road if not for my parents, who instilled within me a love of creative pursuits, science, and language, all of which finds a place in this thesis. To my parents, thank you. My siblings have also been great friends throughout this process. My best friend stayed in touch weekly while completing Pilot Training and deserves my recognition; as do many other friends from Ohio and the Academy.

I thank my classmates for their insights both academically and professionally.

I owe uncountably infinite thanks my Maker.

David A. Ryan

Table of Contents

Page

Abstract iv

Dedication v

Acknowledgments vi

Table of Contents vii

List of Figures xi

List of Tables xiv

List of Acronyms xv

I. Introduction 1

1.1 Protecting the Network 2
1.2 Goal and Objectives 5
1.3 Approach 6
1.4 Thesis Overview 8

II. Background for Flow-Based Intrusion Detection 10

2.1 Network Modeling 10
2.1.1 Model Limitations 11
2.1.2 Network Topology Model 12
2.1.3 Simulated Network Traffic 16
2.1.4 Network Simulation Environment 17
2.1.5 Visualization of Network Model 18
2.1.6 MASON 19
2.2 Pattern Recognition 22
2.2.1 Formating Data 23
2.2.2 Feature Selection 24
2.2.3 Embedded Methods 25
2.2.4 Wrapper Methods 26
2.2.5 Filter Methods 26
2.2.6 Hybrid Methods 28
2.2.7 Classification and Regression 29

Page

2.2.8 Clustering . . . 33
2.2.9 Support Vector Machines . . . 35
2.3 Intrusion Detection . . . 44
2.4 Flow-based Intrusion Detection . . . 45
2.5 Network Attacks of Interest . . . 48
2.5.1 Denial of Service Attacks . . . 48
2.5.2 Vulnerability Scans . . . 50
2.5.3 Worms . . . 52
2.5.4 Man-in-the-Middle and Eavesdropping . . . 53
2.5.5 Trojans and Back-doors . . . 56
2.6 Multi-Objective Optimization . . . 59
2.6.1 Multi-Objective Problems . . . 59
2.6.2 Stochastic Search . . . 60
2.7 Multiagent Systems . . . 64
2.8 Reputation and Trust . . . 68
2.9 Defensive Measures of Network Agents . . . 71
2.10 Summary . . . 75

III. MFIREv3 Design Methodology and Implementation . . . 77

3.1 Intrusion Detection System Formalization . . . 77
3.2 Simulation Environment . . . 78
3.2.1 Network Design . . . 81
3.2.2 Multi Agent System Design . . . 86
3.2.3 Simulated Network Traffic . . . 94
3.2.4 Observations and Features . . . 95
3.2.5 Attack Models . . . 98
3.3 Training the Agents . . . 105
3.3.1 Generating training data . . . 107
3.3.2 Training the Classifier . . . 108
3.3.3 Feature Selection . . . 109
3.3.4 Kernel Method Selection . . . 113
3.3.5 Testing the MAS . . . 115
3.4 Movement models . . . 116
3.4.1 Agents using a reputation model . . . 116
3.4.2 Agents using a Stochastic Model . . . 120
3.4.3 Deterministic Search with Maximum Cover . . . 122
3.5 Defensive Measures Methodology . . . 124
3.6 Visualization . . . 126
3.7 Summary . . . 128

Page

IV. MFIREv3 Experimentation and Analysis of Results . . . 129

4.1 Software Testing . . . 129
4.2 K-Fold Cross-validation . . . 133
4.3 Feature Selection . . . 134
4.4 Kernel Functions . . . 135
4.5 Movement Models Experimental Design . . . 137
4.5.1 MFIREv3 Reputation System Experimental Design . . . 139
4.5.2 MFIREv3 Stochastic Search Experimental Design . . . 139
4.5.3 MFIREv3 Deterministic Maximum Cover Algorithm Experimental Design . . . 141
4.6 Movement Model Analysis . . . 143
4.6.1 MFIREv3 Reputation System Performance Assessment . . . 143
4.6.2 MFIREv3 Stochastic Search Performance Assessment . . . 147
4.6.3 MFIREv3 Deterministic Algorithm Performance Assessment . . . 151
4.7 Defense Analysis . . . 153
4.8 Summary . . . 155

V. MFIREv3 Conclusions and Future Research . . . 156

5.1 Conclusions . . . 156
5.2 Future Research Activity . . . 159
5.3 Applications in Real-Word Settings . . . 160
5.4 Overall Summary . . . 161

Appendix A: History . . . 162

Appendix B: Network Threats . . . 165

Appendix C: Popular DES Engines . . . 169

Appendix D: Popular SVM Packages . . . 170

Appendix E: Intrusion Detection System Details . . . 172

Appendix F: Kernel Tests . . . 174

Appendix G: MFIRE System Details . . . 177

Bibliography . . . 181

Page

Vita 196

List of Figures

Figure Page

2.1 Illustration of LAN Topology [44] 11

2.2 Probability density function for Pareto distribution, $\alpha = 1.0$, $b = 1.0$ 17

2.3 Basic elements of the MASON model and visualization layers [116] 21

2.4 A general perspective of a pattern recognition system [149] 23

2.5 Poor (a) and Optimal (b) separating hyperplanes of an SVM. The poorly separating hyperplane offers bad generalization ability whereas the optimal separating hyperplane perfectly divides both data sets by maximizing the margin of the hyperplane [120] 38

2.6 SVM separating features with a hyperplane in a higher dimensional space [96] . 40

2.7 Margin Bound for Multiclass SVM [185] 42

2.8 Network flow 46

2.9 Taxonomy of DDoS Attack Mechanisms [122] 49

2.10 Illustration of SSL Thwarting MitM Attack 54

2.11 Detection of MitM Attack in UDP Mode [169] 55

2.12 Event Classification Abstraction Levels for Harrier [40] 58

2.13 Generic trust model: conceptual relationships [77] 69

3.1 MFIRE v3 Package Diagram. 79

3.2 MFIREv3 class diagram. 82

3.3 MFIREv3 Activity diagrams for the Agent 87

3.4 MFIRE Activity Diagram Message Exchange 88

3.5 MFIREv3 detailed activity diagrams for the agent provider and the agent 92

3.6 Illustration of Linear Dependecies in Features 97

Figure Page

3.7 Illustration of Non-Linear Separability: Magenta-DDoS Worm- Red Scan-Green MitM-Yellow Trojan-Blue Normal-Black(Covered); Feature 1: Number of distinct destination addresses Feature 2: Total number of inbound bytes Feature 3: Std. Dev-Ratio of packets to (dest addr, dest port) 98

3.8 Illustration of MFIREv3 DDoS 100

3.9 Illustration of MFIREv3 Worm 101

3.10 Illustration of MFIREv3 Scan Report 103

3.11 Illustration of MFIREv3 Man-in-the-Middle Attack 104

3.12 MFIREv3 offline training and online testing execution paths 106

3.13 Add and Del Mutations 110

3.14 Illustration of Non-Linear Separability: Magenta-DDoS Worm- Red Scan-Green MitM-Yellow Trojan-Blue Normal-Black(Covered); Feature 1: Number of distinct destination addresses Feature 2: Total number of inbound bytes Feature 3: Std. Dev-Ratio of packets to (dest addr, dest port) 114

3.15 Kernels in Common Coordinate System [77] 115

3.16 Classification Rule 120

3.17 Movement Actuator 3-feature Color Determines Probability 121

3.18 Worm Rerouting Network MFIREv3 125

3.19 Illustration of MFIREv3 Visual Interface 127

4.1 Illustration of DDoS vs Normal Features F1: Num inbound Bytes F2: Num inbound Packets 137

4.2 Average Agents Moving (8-agent Reputation Model) 143

4.3 4 Agent Reputation Model: Accuracy & False Positives vs. time 144

4.4 8 Agent Reputation Model: Accuracy & False Positives vs. time 145

4.5 Accuracy box plots 146

Figure Page

4.6 Average Agents Moving (8-agent Stochastic Model) 148

4.7 4 Agent Stochastic Model: Accuracy & False Positives vs. time 149

4.8 8 Agent Stochastic Model: Accuracy & False Positives vs. time 150

4.9 Average Agents Moving (8-agent Deterministic Model) 152

4.10 Figures A: Illustrating the effective spread of a DDoS attack without Rate Limiting and B: Illustrating Rate Limiting's effectiveness against DDoS attacks 154

B.1 Short Taxonomy of Attacks [78] 167

F.1 Illustration of DDoS vs Normal Features F1: Num inbound Bytes F2: Num inbound Packets 174

F.2 Illustration of Scan vs Normal Features F1: Num distinct dest addrs F2: Num distinct dest ports 175

F.3 Illustration of Worm vs Normal Features F1: Ratio of packets to dest tuples F2: Ratio of packets from source tuples 175

F.4 Illustration of MitM vs Normal Features F1: Local Num inbound Bytes F2: Ratio of source ports to addr 176

F.5 Illustration of Trojan vs Normal Features F1: Num distinct source ports F2: Num inbound packets 176

G.1 MFIRE: Messages sent by the providers and received by agents 178

G.2 MFIRE: Messages sent by agents and received by the providers 179

G.3 MFIRE: Messages sent by agents to other agents 179

G.4 MFIRE: Messages involved in agent migration 179

List of Tables

Table Page

2.1 Parameters for the spread of active worms [111] 54

3.1 Comparison of MFIRE Iterations 93

3.2 How the Provider Rates Shared Feature Values 119

4.1 Feature Selection Classification Accuracies 135

4.2 Feature Selection Single Attack 136

4.3 Linear Kernel Accuracies 138

4.4 Reputation System Overall Accuracy 146

4.5 Wilcoxon Rank Sum p-values 147

4.6 Stochastic Search Overall Accuracy 150

4.7 Deterministic Search Overall Accuracy 152

List of Acronyms

Acronym	Definition
IDS	Intrusion Detection System
MAS	Multi Agent System
ID	Intrusion Detection
AS	Autonomous System
BGP	Border Gateway Protocol
LAN	Local Area Network
CD	Controlled Distance
FKP	Fabrikant-Koutsoupias-Papadimitriou
HIDS	Host-Based Intrusion Detection System
NIDS	Network Intrusion Detection System
UDP	User Datagram Protocol
SVM	Support Vector Machine
EA	Evolutionary Algorithm
RPC	Remote Procedure Calls
TCP	Transmission Control Protocol
IANA	Internet Assigned Numbers Authority
HTTP	Hyper Text Transfer Protocol
MOEA	Multi-Objective Evolutionary Algorithm

A MULTI AGENT SYSTEM FOR FLOW-BASED INTRUSION DETECTION

I. Introduction

The United Nations' Telecommunication Chief, Hamadoun Toure, stated that the next world war could potentially take place in cyberspace. "Loss of vital networks would quickly cripple any nation, and none is immune to cyber attack" [31]. Cyber attacks taken against the United States, China, South Korea and Estonia illustrate that government networks are vulnerable and it is far more difficult to apprehend attackers in cyberspace [31]. Even if attackers are identified, they are often located in countries with favorable jurisdiction for their network activities. Former Department of Homeland Security Stewart Baker stated that an increasing number of U.S. companies are retaliating against attacks with "strike-back" technology. He also stated that these retaliations could violate state and federal law [24].

Passive defense systems have bolstered network security, however 5.5 billion attacks took place in 2012 according to Symantec; an 81% increase from 2011 [87]. It is clear that cyber attackers are not deterred from infiltrating high value networks such as banks, governments, and military. The reason is simple, the benefit of acquiring sensitive information far outweighs the cost of working to breach the passive security. The key to deterring malicious attacks is decrease the probability of unauthorized access to networks, quickly detect and eliminate threats once they are detected, and most controversially; increase the cost of launching an attack by counter offensive measures [24]. This investigation focuses on the later two approaches for improving network cyber security.

As stated by Symantec security expert, Donna Howell, the key to not being victimized by cybercrime is to maintain systems that are secure in order to deter attackers [87]. All

systems are vulnerable, but attackers target the weakest defenses much like the lion eats the slowest antelope [87]. This may be true in the commercial market, however government systems are targeted despite maintaining the highest levels of network security [136]. For this reason, President Barack Obama issued twelve cyber-security initiatives to bolster network defenses [136]. This research effort develops from the initiative, *"Deploy an intrusion detection system of sensors across the Federal enterprise"* [136].

1.1 Protecting the Network

Network attacks come in various delivery formats and serve multiple purposes. Attacks can generally qualify under a few major classes. A subset of these major classes of attacks allows for good testing for the purposes of grading the abilities of an anomaly based intrusion detection system.

Three major classes of network cyber security threats include [163]:

- Attacks that consume network resources, denying their use for legitimate purposes
- Attacks that infiltrate systems, allowing attackers unauthorized access to system resources, including sensitive data, data storage, privileged relationships with other systems, and network connectivity
- Unauthorized vulnerability scans, providing attackers vital reconnaissance in preparation for infiltrating activities

A quality attack likely falls under one attack class and consist of a series of attacks. A series of attacks allows achieving an overarching goal through mutual reinforcement. For example, a successful scan allows an attacker to infiltrate networks with great stealth and precision; once in control of multiple hosts, the attacker may use them to launch a distributed denial of service attack on another target system or network. Alternatively, the attacker can use these newly acquired assets to conduct further scans more efficiently. As

another example, a clever attacker may launch a denial of service attack on a highly visible service to divert the attention of security personnel from his infiltration activities.

Within these categories, many types of intrusion are recognized and five of them are evaluated in this research effort [126]. The attacks examined in this research effort include: Denial of Service, Worm, Scan, Trojan, and Man-in-the Middle. These represent a list of attacks that create flow-based anomalies in the network in both a local and distributed fashion. This is not a plethoric list of possible attacks, but does list the common categories of attacks on networks. For a list of common network threats see Appendix B.

Intrusion Detection Systems protect the local area network from malicious traffic of the outside Internet. The principle focus of the system is to monitor local traffic for signs of malicious activity. In most Intrusion Detection System (IDS) no effort is expended to *share* the obtained information with another IDS [126]. Compiling information over a broad view of a distributed network from multiple sources provides greater information than is obtainable from a single entity. IDSs with shared network statistics can better detect threats progressing across network nodes [77].

Threats are unpredictable in their location, but it is beneficial to move an observing Intrusion Detection Systems to the point of attack to improve classification. To perform this task, we explore the concept of multi-agent IDSs, in which individual agents are able to move throughout a distributed network, and share data to collectively determine if an attack is occurring.

Multi Agent System (MAS) for Intrusion Detection (ID) is not a new concept. Multi Agent Systems are appropriate for highly dynamic environments and environments with distributions of data, expertise, and location [81]. Networks are often complex, dynamic environments with security requiring various expertise.

The pursuit of the research is to improve the current reputation-based Multi-Agent Intrusion Detections System, MFIRE, developed by David Hancock [77] and Timothy

Wilson [185]. The two common characterizations of an IDS are network or host-based and signature-based or behavioral-based. All characterizations have fatal flaws in classification capability.

A signature-based IDS reacts to strings flowing across its detection system that match a repository of malicious signatures [42]. This system is effective, yet it requires an up to date repository to detect many attacks. An up to date signature repository can also become overwhelmed comparing each signature to the signatures in an ever expanding repository. A behavior-based IDS reacts to system activities considered anomalous [112]. This means that the system must already be infected before the IDS reacts. The technique integrated in this effort comprises of a combination of the two IDSs. Agents monitor inbound and outbound traffic flows in a fashion similar to that of a signature-based detection method, however it does not keep a repository of malicious signatures. Instead, the agents examine statistics related to these traffic flows and react if a disturbance to the normal statistics occurs.

For the best results, agents need to use the best available features to classify the traffic flows. Since anomalous statistics indicate specific attacks, the use of features that provide the same statistics regardless of the state of the network work against accurate classifications. We use a feature selection Multi-Objective Algorithm to select features that indicate the state of the network. In that sense, as opposed to maintaining an ever-growing list of malicious signatures, agents maintain a list of "universal" features to classify attack *types* instead of *exact/known* attacks.

Network-based intrusion detection typically occurs at the network gateway. The IDS has an overarching view of the entire network and is capable of detecting disturbances between groups of hosts. The network-based IDS however, fails to protect individual hosts from attack [167]. Detection of an attack on a single host effectively requires a host-based IDS. The difficulty with host-based intrusion detection is the inability to recognize

malicious activity that spreads across the entirety of the network [36]. Previous work on mobile agents by Hancock [77] and Wilson [185] aimed to combine the benefits of both network and host-based intrusion detection. Multiple autonomous mobile agents reside transiently at network hosts. The agents are capable then of addressing local anomalies and summarizing the local information for a central entity performing network based intrusion detection functions. This effort is further extended by the elimination of a single central controller for agent movement and network-based intrusion detection. Instead, the agents themselves each keep track of their own local statistics as well as a combined set of summarized statistic in order to recognize both local and network based anomalies.

Despite the best efforts of network security, intrusions occur. Once they occur agents are capable of taking an active roll in limiting the damage, spread, and overall effectiveness of an attack. Limiting the spread of a Denial of Service attack can be as simple as limiting the traffic rate within the network. Closing ports on an infected node from a virus or worm effectively removes the node from the network preventing the spread to uninfected systems.

1.2 Goal and Objectives

The *goal* of this research is to develop a scalable distributed Multi Agent System (MAS) for the defense of flow based system attacks and anomaly detection and identification.

We achieve this cyber security goal and increase the effectiveness of a flow-based, multi agent network attack classifier with the following high-level *objectives*:

- Continue design and evaluate a multi-agent intrusion detection system using a Reputation system
- Evaluate the MFIREv2 multi-agent intrusion detection system using stochastic search

- Design and evaluate a multi-agent intrusion detection system using deterministic search with search incentives and Maximum Cover
- Design and evaluate a Multi Objective Evolutionary Algorithm for best subset feature selection
- Determine if attacks can be classified using a Linear Kernel as opposed to the Radial Basis Function when using MOEA selected features
- Create a robust distributed simulation framework for evolving self organizing multi-agent systems
- Create a robust simulation framework for the automated defense of the network

Research results include an evaluation of the environment's classification performance. The research produces an effective and efficient simulation environment to conduct ongoing, flow-based intrusion defense experiments.

1.3 Approach

This research effort continues a framework for conducting simulations of networks under attack, and a multiagent system which is trained to detect threats and provide defensive measures. Feature selection and training take place outside the MFIRE system. Any system can be used for these processes with the effectiveness validated within the MFIRE system.

The multi-agent framework provides an ongoing platform for MAS and network research. The agents aim to find better node locations for classifying an attack, however the user can instantiate the agents to perform other actions as well. We seek to compare the effectiveness of three distinct models for attack identification. These models use the same classifier, but agent movement decisions follow three different systems. A reputation system is used among multiple agents called *providers* to dictate agents' movement

decisions. With the second model, the agents are allowed to move freely on their own, with their behavior optimized using a genetic algorithm. Finally, a more direct search is implemented using a deterministic search method optimized for Maximum Cover of the network. For a comparison, we also examine the case when agents are at a fixed random location and no movement is allowed called the *baseline* case.

This research follows from three previous efforts by Eric Holloway [84], David Hancock [77], and Tim Wilson [185]. The three previous efforts introduce similar concepts that can envelop a broader Intrusion Detection System. We continue previous MFIRE developments which tests the effectiveness of a flow-based, multi-agent network attack classifier by moving agents to improved locations. We also seek to include work inspired by Eric Holloway that created automated defense of the network by working to stop attacks without user intervention being necessary.

Included in this effort is thorough feature selection testing. None of the previous efforts placed significant emphasis on the quality of features used by agents in attack classification. The expanse of all network attacks be tested on an "optimal" feature subset determined by an Evolutionary algorithm. Also the effect of placing the feature set for best classifying a specific attack when only that attack and normal traffic flow take place is examined.

Wilson [185] took elements of both Holloway's [84] and Hancock's [77] efforts to create a single integrated simulation environment for multi-agent flow-based intrusion detection. Wilson's effort included executing initial baseline experiments to test the combined approach, and demonstrate the potential usefulness of such an environment.

This research effort aims to improve upon the three efforts. The free-movement and reputation models are compared to a baseline, but the reputation model no longer requires a centralized controller. Making the agents completely autonomous and self-organized eliminates the *single point of failure* while maintaining the algorithmic approach of the reputation model. We aim to take the automated defense approach introduced by

Holloway and apply it to attacks outside of the scope of his effort. The deterministic approach introduces the idea of node memory for comparison with the other models. Node memory allows agents to avoid "bad" nodes, but increases the amount of data in the system dramatically.

1.4 Thesis Overview

This chapter provides an introduction to the problem with goals and objectives to approach a solution. Chapter 2 explores the various background concepts in approaching a Multi-agent system for intrusion detection. Chapter 3 details the design and implementation of MFIREv3. Chapter 4 presents the testing and analysis of the results of our system. Chapter 5 concludes with a summary of the effective results of the research and the overall impact of the tests. Opportunities to further investigate these impacts is proposed as future work in the conclusions of Chapter 5.

The Appendix includes a history of the MFIRE design from previous research efforts. It also includes fine grained details of system communications, tests, and lists of popular products relating to ID that are not used in this effort. The Appendix is referenced at points in the document for related information. Portions of the document, including the Appendix, reference taxonomies of attacks, IDS packages, and Classification packages. These taxonomies help establish the decision process of choosing a subset of the taxonomy, as well as shed light on the other options.

Results from the experiments demonstrate that MFIREv3's movement models allow the agents to find nodes in the network that increase the system's classification accuracy. The original *decentralized* approach proposed by Wilson [185], is thoroughly tested for the first time and provides good vantage points using fewer network resources than the other two movement models. The deterministic search model provides a new avenue of development involving specific node "reputation". Tests illustrate an improvement in classification accuracy using this approach over the baseline model.

The results also demonstrate the effectiveness of "optimal" feature sets for classifying attacks. Both in the cases where the agents needed the optimal set to detect all five attacks and just one of the five attacks, accuracy increased from the random feature selection used in MFIREv2. With respect to the defensive measures, the results illustrated that agents are effective in limiting the effects of attacks.

MFIREv3 can be used for any number of experiments involving flow-based network simulation or multi agent system. The MFIREv3 simulation environment provides a scalable framework to execute network threat analysis using robust multi-agent, flow-based techniques. Conducting a wide variety of experiments, not specific to intrusion detection, is possible with the current framework.

II. Background for Flow-Based Intrusion Detection

This investigation focuses on the improvement and expansion of network-based attack recognition. The primary effort focuses on determining an "optimal" feature set for recognizing a flow-based attack, the classification of these attacks on a system, autonomously determining and finding "optimal" agent locations for detection, and the mitigation of flow-based network attacks once they take place. This chapter supports these goals by discussing the foundations, concepts, and current research relevant to network intrusion detection, classification, feature selection, and defense.

Section 2.1 discusses the various aspects of network modeling including traffic, topology, and visualization. Section 2.2 details the important aspects of pattern recognition, which is the principal component of anomaly-based intrusion detection systems. The applications of these pattern recognition techniques is discussed in section 2.3. Section 2.5 introduces the five attacks implemented in this research effort. Multi-Objective Optimization is discussed in section 2.6 with the applications of a Multi-Agent System in section 2.7. The chapter concludes with a discussion of reputation in a Multi-Agent System, Defensive measures of agents.

2.1 Network Modeling

Network topology is the arrangement of various nodes within a computer network. There is both a physical and logical topological structure, where the physical model illustrates the locations of the nodes and their interconnections while the logical model illustrates the data-flow between components within the network. A collection of routers and other networked devices under the same administrative control is called an Autonomous System (AS), and gateway routers are responsible for forwarding traffic to and from other autonomous systems. An example of an AS is a Local Area Network (LAN),

where any given node within the LAN has physical links to other nodes within the network that pass data to one another. The Internet consists of many AS and other sub-networks. An inter-autonomous system routing called Border Gateway Protocol (BGP) handles traffic between AS [104]. Figure 2.1 illustrates a basic Local Area Network topology where traffic would flow between the separate nodes and to the Internet.

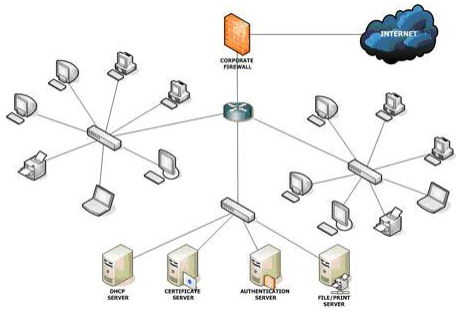

Figure 2.1: Illustration of LAN Topology [44]

Two primary modeling concepts within this domain are *topology* and *traffic*. With a firm grasp of these principle aspects, we can devise a network modeling system that achieves a good balance between efficiency and accuracy. Testing and evaluation must be carried out on a modeled network. It is not appropriate to deploy an untested defensive application without evaluation on a modeled network. Testing and evaluation on a modeled network allows for controlled conditions over a range of situations [17]. Modeled networks also provide reduction of maintenance that would otherwise be unwieldy on an operational environment.

2.1.1 Model Limitations.

Realistic modeling of Internet Network Topology is difficult given the overall organization of the complex network itself. The internet is a network of nodes that are themselves networks. Certain entities represent specific qualitative behaviors within the simulation. A *qualitative simulation* as defined by Kuipers [102], is “the prediction

of the possible behaviors consistent with incomplete knowledge of the structure of the physical system." These qualitative behaviors that require representation in this research investigation include the actions of both malicious and benign users of the Internet, as well as development of a network topology given the requirements of the system.

A qualitative behavior or solution can be either *sound* or *complete*. A sound solution is one where no trajectory which is the solution of a concrete equation matching the input can be missing from the output. A complete solution is one that does not produce any false prediction for any particular input. This is in accordance with the incompleteness of any strong formal system as proven by Kurt Gödel [56].

Despite this limitation, many refined models exist capable of handling all of most practical data. The modeling process illustrated in this effort is inspired by a process of starting with a simple model and refining the model in steps until arriving at the intended degree of "realism" of the system [82].

2.1.2 Network Topology Model.

The creation of a valuable environment for performance evaluation is the direct product of network topology Yook et al [188]. It is often the case that protocols working seamlessly on prototypes fail to scale up to real networks making them ineffective. This is caused by the ineffective realization of key components of real network complexity in a model. The key and greatest challenge of modeling network topology is capturing the dynamic properties of a real network with topologies growing and changing over time. Effective dynamic models allow defensive measures to be tested on realistic predictions of the future network topology [188].

Early attempts at topology modeling relied heavily on random graphs [28, 52]. One of the most popular network models was the Waxman Model, which created graphs based on the Euclidean distance between nodes [180]. Although effective modeling small networks,

the ability to model complex network topology fell short with Waxman's model requiring a new approach.

The next advancement in modeling network topologies proposed the induction of each different hierarchical level of the network model [49]. The Waxman model was never intended as an all-purpose model for network topology, as Waxman designed the model as a test for Minimum Steiner Tree Problems [35]. The difference is that intranetwork data and internetwork data are not the same and need to be treated differently. This effort lead to the creation of two network topology generators; Tiers [49] and Transit-Stub [97, 190].

In the late 1990's research conducted on internet topology found that the Internet is a scale-free network: a network whose degree distribution follows a power-law, at least asymptotically [55]. That is, the fraction *P(k)* of nodes in the network having k connections to other nodes goes for large values of k as

$$P(k)\ ck^{-\lambda} \tag{2.1}$$

With this discovery, three eigenvalues were calculated corresponding to the power-law that characterize the inter-domain topology measurements derived from BGP routing tables. The three values were the Rank exponent, Outdegree exponent, and the Eigen exponent. The calculations of these exponents lead to the discovery that they are all practically equal, meaning that the Internet could be classified as a scale-free network [54].

Utilizing the findings of the scale-free network model, Barbási and Albert incorporated the results into a study of the attack tolerance of complex networks [4, 16, 45]. They concluded that the Internet's AS-level architecture follows a power-law distribution, that it is resilient to random attacks but very vulnerable to attacks targeting central important nodes. These select nodes have most of the links to other nodes, while most of the nodes have very few links making most of the nodes resilient to attacks with the most important ones very vulnerable. Barbási later worked with Yook in further demonstrating

the shortfalls of existing topological network models and refining what came to be known as Barbási-Albert (BA) models [188].

Although certain critics object to the power-law approach, specifically Willinger, it can be concluded from the BGP-derived AS maps are "Pareto-type principles; that is, a small number of nodes have many neighbors, while most nodes are connected to only a small number of neighbors" [101]. For more on Willinger's opposition to the BA model see [101, 183]

Willinger instead advocates for using domain knowledge and exploiting the details that matter when dealing with a highly engineered system [183]. In an interesting development using this approach and power-law models, a model known as Fabrikant-Koutsoupias-Papadimitriou (FKP) developed [54]. FKP successfully generated topologies exhibiting the power-law relationships, but instead of using a purely stochastic approach, these power-law properties arise from a simple Multi-Objective Optimization (MOO), involving "last-mile" connection costs and transmission delays as measured in hops [161]. Each node i arrives at a uniformly random point and attaches itself to the node j that minimizes the weighted sum :

$$min_{j<i}\{\alpha \cdot d_{ij} + ecc(j)\} \tag{2.2}$$

The Euclidean distance between the nodes, d_{ij}, represents the "last-mile cost." The relative importance of this objective is determined by the weight, α. The other term is the *eccentricity* of j and captures the distance from the center to j.

Spatharis et al. present a well-balanced approach called the Controlled Distance (CD) Model [161] based on the power-law models of *FPK* [54] and Willinger model [183].

The objective of CD is to address the need for edges between nodes that are not quite leaves, nor particularly central, but are of intermediate centrality. As each node i is added

to the network and linked to the node j, a second edge is attached from j to another node k minimizing

$$min_k\{\alpha \cdot d_{jk} + ecc(k)\} \tag{2.3}$$

over all k such that the hop distance from j to k is at most a constant c [161].

By decreasing the power-law exponent while having high average degree and several leaves, this model is considered a top performing topological model [161]. By achieving similarity to the Internet's AS graph, this model allows for extensive testing and evaluation with improved scalability for protocols.

Topology generators include:

- TopGen [161]
- Tiers [49]
- GT-ITM - Georgia Tech Internetwork Topology Models [35]
- Inet [186]
- nem [119]
- BRITE [125]
- GDTANG - Geographic Directed Tel Aviv University Network Generator [15]
- RealNet [41], [40]

RealNet is a newer addition to the list [77] which relies on publicly available datasets including BGP tables and traceroute records, as does [55]. It addresses some of the problems inherent in these datasets and does not attempt to fit specific power-law-based statistics. For example, it gives direct consideration to the IP-aliasing problem, whereby

more routers may be inferred than actually exist because each router has a different IP address for each of its interfaces. It also factors in likely policy relationships between neighboring autonomous systems [77].

In summary, many shortfalls exist with the power-law approach to modeling network topology even with the Internet exhibiting Pareto-type principals. The ideal topology modeling approach should appease both the engineers responsible for implementing actual Internet topologies, as well as the protocol and application developers seeking validation that the program works for an extended time frame. Although this perfect topology modeling framework does not exist, FKP/CD provides a reasonable approach [185].

2.1.3 Simulated Network Traffic.

In addition to modeling network topology, the simulated Internet traffic must be realistic in order to properly examine protocol. Traffic modeling began with simple Poisson distributions, but now includes models that exhibit self-similarity [20]. Even more so than Internet network topology, the traffic created between nodes has been shown to display scale-invariant statistics.

A widely used model for packet routing is the Poisson model. Willinger and Paxson [184] advocate against the Poisson model for better, more fractal-like traffic distribution models [77]. For network traffic, a Pareto model may be preferred.

The Pareto model exhibits scale-invariant behavior [69]. It has a density function $P(X) = \frac{\alpha b^{\alpha}}{x^{\alpha+1}}$, $x \geq b$, which has a heavy tail [99]. Figure 2.2 shows an example for $\alpha = 1.0$, $b = 1.0$. Willinger and Paxson explain that this heavy tail accounts for the fractal nature of aggregated network traffic [184]. To generate a random Pareto-distributed sample, inverse transform sampling is used. Given a random variable U drawn from the uniform distribution $(0, 1)$, T, is Pareto-distributed [48], and given by

$$T = \frac{b}{U^{\frac{1}{\alpha}}} \tag{2.4}$$

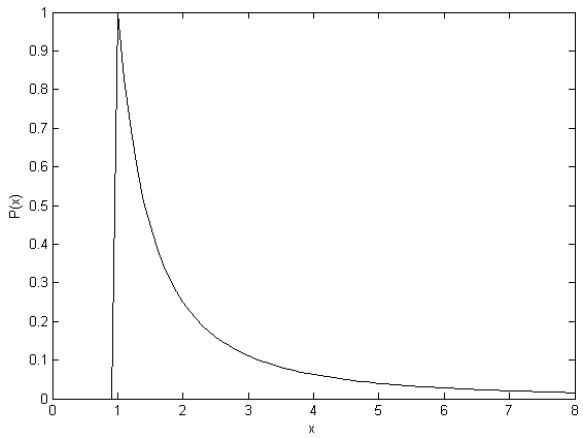

Figure 2.2: Probability density function for Pareto distribution, $\alpha = 1.0$, $b = 1.0$

2.1.4 Network Simulation Environment.

This section focuses on the underlying network simulation framework. Specifically, we discuss Discrete Event Simulation. This method views the simulation as being composed of a chronological sequence of events, each of which occurs in an instant and changes the state in the system, possibly resulting in more events being scheduled. Comprehensive treatment of Discrete Event Simulation is given in [13].

Components of DES systems include:

- Clock - The simulation keeps track of current simulation time in appropriate measurement units, but unlike in real time simulations, time in a DES jumps from one instantaneous event to the next.
- Schedule - The set of events to handle, typically implemented as a priority queue sorted by event time.
- Random-Number Generator - pseudorandom, which is desired in order to support a rerun of a simulation with exactly the same behavior
- Streams - Independent function streams for each function variable

Typical usage of a DES includes the gathering of statistics, for which facilities may be provided, and the specification of a stopping condition. As may be the case with continuous- but not real-time simulation, a discrete event simulation runs at a rate that is not tied to the real-world clock. When resources permit, simulations may be run potentially much faster than real time, which is useful for collecting large amounts of statistics. In other cases, it may be desired that simulations run much slower than real time, perhaps paused for an extensive period of time via checkpointing, which is useful for direct observation and analysis of system dynamics.

A recent, innovative network-based anomaly detection system is presented in [93]. The authors use a two-stage classification approach to detect novel intrusions of various types, and is shown to have good empirical performance. Many IDS systems have shown in recent years to achieve good performance with real-world traffic [19, 80, 81, 85, 90, 129, 187]. Our approach is substantially different, in that we seek a robust environment to generate *simulated* network traffic; and the goal of our research focuses on *improvement* in performance given the movement of agents, not on achieving absolute performance. Thus it is difficult to find existing systems to compare our approach, however many of systems previously mentioned provide a basis for our key concepts of multi-agent systems, network-based detection, anomaly-based classification, and flow-based statistics.

One final method of performance enhancement includes parallelization of the DES. As is the case with any system, parallelization allows for network resources to be used more efficiently by reducing bottlenecks in computation. Parallelization of DES is discussed extensively in [64]. More recently, Park and Fishwick present their work using graphics processing unit-based clusters in [139].

2.1.5 Visualization of Network Model.

In this thesis investigation several expressive visualization techniques for intrusion detection and anomaly data are considered. The key idea is to classify the underlying

data according to its prominence on the resulting visualization by importance value. The importance property drives the visualization pipeline to emphasize the most prominent features and to suppress the less relevant ones. The suppression can be realized globally, so the whole object is suppressed, or locally. A local modulation generates cut-away and ghosted views because the suppression of less relevant features occurs only on the part where the occlusion of more important features appears [60].

Features within the data are classified according to a new dimension denoted as *object importance*. This property determines which structures should be readily discernible and which structures are less important. Next, for each feature various representations (levels of sparseness) from a dense to a sparse depiction are defined. Levels of sparseness define a spectrum of optical properties or rendering styles. The resulting image is generated by ray-casting and combining the intersected features proportional to their importance. An additional step to traditional volume rendering evaluates the areas of occlusion and assigns a particular level of sparseness. This step is denoted as importance compositing. Advanced schemes for importance compositing determine the resulting visibility of features. If the resulting visibility distribution does not correspond to the importance distribution, different levels of sparseness are selected.

The applicability of importance-driven visualization is demonstrated on several examples from medical diagnostics scenarios, flow visualization, and interactive illustrative visualization in [175]. Importance-Driven Feature Enhancement propels the utilization of the most vital information with limited noise from extraneous data or applications. The visualization of the network model must cater to the needs of the research effort. In this investigation, the agent based system must effectively demonstrate the actions of the agents.

2.1.6 MASON.

There is a wide variety of discrete event simulation engines available for building an agent based simulation (see Appendix C). Typically, each emphasizes a certain domain or

technique. For example, OMNeT++ [174] is geared toward network simulation, as is ns2 [121] and cnet [122, 123], while Parsec's emphasis is parallelism [12]. Finally, MASON [116] caters to the needs of multi agent systems simulation. MASON is used in MFIREv3 as it was in previous versions of MFIRE [77, 185] and in SOMAS [84].

MASON is a single-process DES core and visualization toolkit written in Java [116]. It is flexible enough that it can be used for a wide range of simulations, but emphasizes support for "swarm" simulations with up to millions of agents. It is fast and portable and produces guaranteed replicable results, courtesy of checkpointing facilities.

There are three principle reasons for the selection of MASON as the underlying DES engine:

- This research concerns a multi agent system - MASON's structural expertise
- MASON does not impose nor even provide a predefined abstraction of real-world computer networks. Our implemented network simulation is customized to focus on prototyping a system able to function in a moderately complex network environment. Several of the discrete event simulation engines mentioned in this section are heavily invested in accurate simulation of real-world protocols, network devices, and applications, but the much higher complexity of these environments introduces networking issues not directly relevant to MFIRE's initial implementation.
- We have some prior experience with MASON. This research began as a way to complement the attack mitigation capabilities of Holloway's Self Organized Multi Agent Swarms(SOMAS) [84], and continued through the two previous iterations of MFIRE [77, 185]. SOMAS and all MFIRE versions operate in a MASON-based network simulation.

MASON is designed to handle large numbers of agents in complex environments [116], but it is not explicitly an agent framework, such as the frameworks provided by

JACK, Cougaar, JADE, and others [113]. These frameworks in some cases (e.g. JACK [86] and JADE [21]) provide compliance with the interoperability standard called the Foundation for Intelligent Physical Agents (FIPA) [22]. Such agent frameworks should be explored for use by MFIRE in future research.

The underlying model runs in a layer independent of the visualization layer. Thus, while the visualization facilities enable easy interaction with simulations, simulations may run without visualization, or the visualization can be changed at will, perhaps according to the preferences of the observer. The basic elements of the MASON model and visualization layers are presented in Figure 2.3.

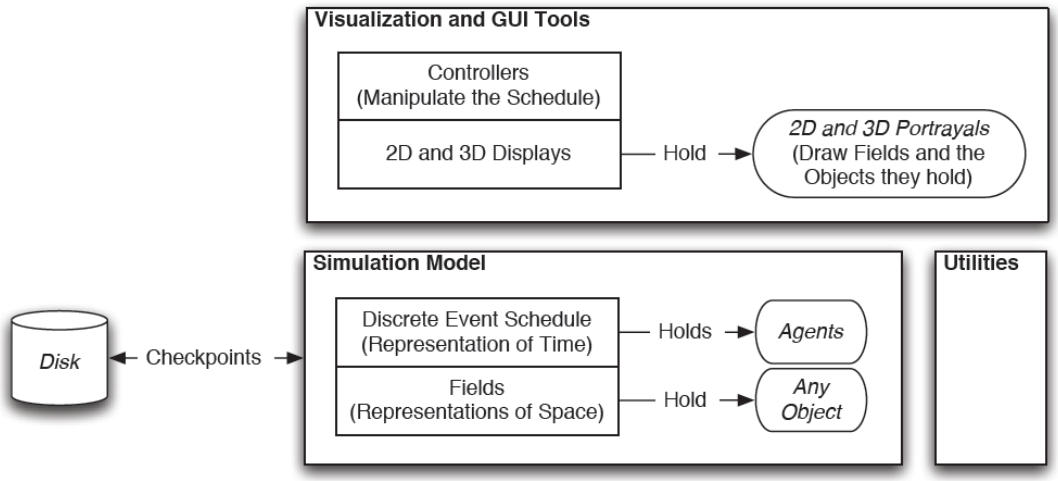

Figure 2.3: Basic elements of the MASON model and visualization layers [116]

The visualization layer runs on top of the simulation model controlling the agents, topology, traffic, and communicating with the Disk. The Disk contains the necessary data producing the simulation and stores the results of the experiment [116].

MASON has been used successfully for applications ranging from physics demonstrations to cooperative target observation in unmanned aerial vehicles to the testing of ant foraging algorithms [116].

For this research, the MASON DES continues to be selected due to its agent-based features and tight Java integration. OPNET, OMNet++ and NS2/NS3 simulate routing at a much more detailed level than is needed for our purposes, though may be explored in future research. The purpose of this research investigation requires the simulation of movement models to test multiple movement algorithms. MASON allows for generic testing of the effectiveness of movement models, classification, multiple feature sets, and actions as proposed by Holloway [84] without the over-detailed framework necessary for packaging the system for commercial use. The MASON DES provides scalability to adapt to changing criterion without a burdensome overhaul of the framework that is required in more detailed environments [116]. OPNET, OMNet++ and NS2/NS3 may provide suitable transitions to a real network implementation.

2.2 Pattern Recognition

Given the groundwork for a simulation framework reflecting the domain of the Internet, we continue by presenting the research in recognizing malicious activity on a network. Pattern recognition is the assignment of a label to a given input value [27]. One example of pattern recognition is classification. Classification works by assigning each input value to one of a given set of classes [153]. If the classification is accurate, precise, and timely, the malicious activity may be counteracted. Figure 2.4 illustrates the general components of a Pattern Recognition System that are discussed in this chapter.

Legitimate traffic has been characterized as presenting short periods of high density traffic followed by long periods of idleness. Deviations from this traffic flow may reveal the presence of anomalous and potentially malicious traffic. Using techniques from the

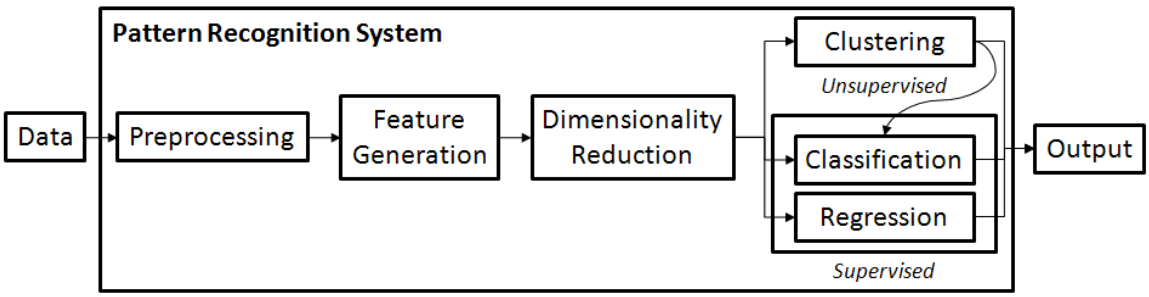

Figure 2.4: A general perspective of a pattern recognition system [149]

broad field of *pattern recognition*, one can classify the traffic features in order to discover intrusions of the network.

Pattern recognition algorithms generally aim to provide a reasonable answer for all possible inputs by matching inputs to what pattern best-fits the observable data [153]. This is very different from pattern matching algorithms, which look for exact matches in the input with preexisting patterns [50].

Authoritative texts on pattern recognition include [50], [79], and [27]. A general description of a pattern recognition system is shown in Fig. 2.4. The definition found in [79] is the primary source for pattern recognition throughout this investigation.

2.2.1 Formating Data.

Preprocessing formats the data, possibly performing some filtering in the case of noise or some system or environmental anomaly. Data preprocessing is an important step in the pattern recognition process. Loosely controlled data gathering methods often lead to impossible combinations (e.g. Gender: Male, Pregnant: Yes), missing values, out-of-range values, etc [10]. Without data preprocessing, this data would cause misleading results and waste resources solving impossible data structures [79].

Feature generation, sometimes referred to as *feature extraction*, is the transformation of raw data into derived data points that may facilitate the characterization of the observed process [14]. When the input data to an algorithm is too large for processing and suspected

to contain redundant data, then the input data is transformed into a reduced set of *features* or *vectors*. The features are selected by the *quality* of information the feature provides in helping to determine the classification of the state [172]. This reduction to a set of features is known as *feature extraction*, and this dimensionality reduction can also be applied to a set of features to further reduce the magnitude of data [173]. This process is known as *feature selection*, and it is further discussed in 2.2.2.

It is often the case that the raw data itself is used for feature measurements. Commonly, features are statistical measurements of the raw data or may be the result of passing the raw data through a mathematical transformation such as a Fourier transform. The results of the transformation create features; statistical data descriptors that distinguish clusters or states. The feature is represented as X. If X is a feature, it has p elements, and components are accessed via subscripts X_j [173].

2.2.2 Feature Selection.

Dimensionality reduction from Figure 2.4, or *feature selection*, filters the available features with the premise that not all features are useful [100]. Some features might be redundant or be harmful for the classification process due to their random nature or tendency to indicate an incorrect class. Even if all features are useful, resource limitations, in terms of computation, bandwidth, or storage, may require filtering the least beneficial information prior to performing clustering, classification, or regression. Feature selection benefits classification by speeding up the learning process, enhancing the generalization capability, improving model interpretation, and potentially reducing the need to enter higher dimensional space in order to separate classes [100].

As noted by Gates at al. in [72], the three principle objectives of feature selection are: 1) improving prediction performance; 2) enabling faster, more efficient prediction; and 3) providing a better understanding of the underlying process that generated the data.

One of the primary reasons feature selection has the potential to greatly improve prediction performance is that it directly confronts the *curse of dimensionality* [23]. Hastie et al. [79] examine some of the many manifestations of this problem. Essentially, such manifestations arise from the fact that in order to maintain the same sampling density enjoyed in a lower dimension, the number of samples must increase exponentially as one moves to higher dimensions. Usually, the number of samples practically attainable is far fewer than necessary to maintain the desired sampling density. The impact of this is either poor performance in classification accuracy or the creation of an artifact and slow rates of computation due to an overabundance of data [79].

Both simple and complex methodical approaches exist for feature selection algorithms. Feature selection methods can be decomposed into two categories: *feature ranking* and *subset selection* [72]. Feature ranking ranks the features by a metric and eliminates the worst performing features. Subset selection searches the set of possible features for the optimum subset, although this is impractical for large numbers as the problem is NP-Hard [5], therefore finding a "good" subset becomes the objective. For this investigation a simple feature ranking approach called ReliefF is examined as well as a more robust subset selection algorithm that also employs feature ranking techniques to reduce complexity.

2.2.3 Embedded Methods.

Subset selection of features can be decomposed into three types of algorithms: wrappers, filters, and embedded [105].

With embedded methods the structure of the class of functions under consideration plays a crucial role, and techniques are developed that are specific to certain classifiers. See [105]. Embedded techniques are embedded in and specific to a model and therefore are not used in this research effort.

2.2.4 Wrapper Methods.

Wrapper methods test feature subsets against the chosen learning machine, which is regarded as a black box [98]. The primary issue becomes determining how to search the space of all possible variable subsets which could take an impossible amount of time [5]. Another issue is assessing the prediction performance of a learning machine to guide the search and halt it once a good fit is reached [98].

2.2.5 Filter Methods.

A filter method typically involves some notion of *feature ranking* independent of the choice of the predictor. This is computationally efficient because it requires only the computation of p scores and sorting the scores [72]. It introduces bias but may have considerably less variance compared to other methods and is therefore robust against over fitting [72, 79]. Filter methods include analyzing performance as a single variable classifier and information theoretic ranking criteria [79].

These filter techniques can be useful but also incur limitations. The underlying assumption is that variable dependencies can be ignored, but in practice, this is not always the case [72]

Methods that score variables individually and independently of each other cannot to determine which combination of variables would give the best performance. Filter methods often provide reasonable performance though, and the computational efficiency is unmatched.

A common technique for filtering features involves ranking them according to how well they separate sample data distributions collected from two classes[72]. The Bhattacharyya distance [26], named after the mathematician, is a measure of the distance between two sample distributions. If two sets of samples are produced by the same process, the estimated distributions should be very close, and the Bhattacharyya distance near zero.

Formally, for discrete probability distributions p and q over the same domain X, the estimate of the Bhattacharyya distance $D_B(p, q)$ is [72]:

$$D_B(p, q) \triangleq -\ln(BC(p, q)) \tag{2.5}$$

where

$$BC(p, q) \triangleq \sum_{x \in X} \sqrt{p(x)q(x)} \tag{2.6}$$

is the *Bhattacharyya coefficient*. For each value of $x \in X$ found in both sample sets p and q, $BC(p, q)$ increases, up to a maximum of 1 when $p(x) = q(x)$ for all $x \in X$; in this case, $D_B(p, q) = 0$ [72]. Conversely, as the limit of $BC(p, q)$ approaches 0, observe that $D_B(p, q)$ increases without bound [72].

Filter methods are capable of *supervised learning* [72]. Supervised learning is the task of inferring a function from labeled training data [79]. A supervised learning algorithm analyzes the training data and produces an inferred function, which is called a classifier. The inferred function should predict the correct output value for any valid input object. This requires the learning algorithm to generalize from the training data to classify the current data input [72].

Relief is a feature ranking method that provides supervised learning by using the nearest in-class H and out-of-class M instance and determines the ranking based on a weighting scale [79]. By picking an instance R and determining that instance's nearest hit and miss it is able to update the weights of its attributes A using the equation [79],

$$W[A] = W[A] - DIFF(A, R, H)/n + DIFF(A, R, M)/n \tag{2.7}$$

where n is the number of instances selected and $DIFF$ is the difference function used to calculate the difference between values of attributes [72].

The difference function is normalized by n, such that all values fall within the range of -1 to 1, with 0 meaning that the two values are exactly the same [4]. The difference

between instances is the total difference, so that using the Manhattan rule the sum of the distances over all attributes provides the difference between classes.

After randomly selecting n instances, the ReliefF algorithm finds the nearest K neighbors from its class as well as the nearest K misses from each of the other classes. This means the weight from each of the other classes is determined by that class's size [79]. This allows ReliefF to work with multi-class data as opposed to datasets containing only two classes.

ReliefF is a very fast method for feature selection as it is a first-order classifier, however it does not take into account feature relationships [72]. In many cases, combinations of less quality features provide a better classifier than a combination of the top features. This is due to the Synergy of features; where the whole is greater than the sum of its parts [74]. In order to find the best set of features for classification, one must use a subset selection approach as discussed in 2.2.9.

2.2.6 Hybrid Methods.

Wrappers and filter methods both use a search algorithm to search through the space of possible features and evaluate each subset. For this effort, a hybrid combination of a filter and wrapper method is considered in order to prevent over fitting and reducing the complexity of performing an exhaustive search on a model. This still benefits from the tailoring to a model without solely relying on a simpler filter.

Memetic frameworks for the hybridization of wrapper and filter feature selection methods are a growing field in classification problems for both single and multiple objective instances [152]. A *meme* is a "unit of culture" which in this case represents the pattern of behavior of the data [46]. Thus a *memetic* framework incorporates the *culture* or *history* of the features.

The purpose of the research effort is to incorporate traditional evolutionary algorithms to improve classification performance while accelerating search for optimal feature subsets.

Single objective feature selection classification using MOEAs is shown to speedup classification of optimal feature subsets [166], while multi-objective/ multi-class problems require the classification of features for multiple classes. In the multi-objective case, certain features that can be shown to improve the classification for one class often degrade the classification capability for another class [27].

The feature selection algorithm cannot select one set of features that optimally classifies for each of the classes. Instead of choosing one subset of features, the MOEA moves towards an optimal front of non-dominated solutions [27]. Each point on the non-dominated front illustrates a selection of features and their relative accuracies within the four classes. Some features that might independently be good features might cause too much noise in combination with other features. On the other hand, a feature that might not help classify attacks by itself could be very helpful for another set of features. The objective of the MOEA is to mutate the combination of features until it finds the best choices for subsets of features. For a discussion of MOEAs, see 2.6.2.

2.2.7 Classification and Regression.

Classification, clustering, and regression represent the three fundamental problems of pattern recognition [27], one or more of which must be addressed by any pattern recognition system.

Classification is a process that assigns one of a discrete number of labels to each input [27]. A is the 'true' output and takes values from the set $\mathcal{A}$. The classifier is $\hat{A}$ and should also take values from $\mathcal{A}$. *Regression* seeks to model a continuous process [100]. The output of the function being modeled is denoted as Y and takes values from some continuous set, such as $\mathbb{R}$, and a predictor for Y is $\hat{Y}$ [79].

In pattern recognition, we want to learn $x \mapsto y$ where $x \in X$ is an object and $y \in Y$ is a class label [100]. In the case of Network Intrusion Detection, x would represent and

individual data stream and y would represent the type of attack or lack of attack of the data stream.

Classification and regression as supervised learning techniques require training data in which inputs are associated with known output [172]. Based on the specific classification or regression technique selected by the system designer, the system derives the necessary parameter values for a process that reliably transforms the training input into the desired output.

Formally, given a training set $(x_1, y_1)...(x_m, y_m)$, we want to train the classifier to generalize such that given a previously seen $x \in X$ it finds a suitable $y \in Y$. In other words, we want to find a classifier $y = f(x, \alpha)$ where α are the parameters of the function [100].

We can attempt to learn $f(x, \alpha)$ by choosing a function that performs well on training data:

$$R_{emp}(\alpha) = 1/m \sum_{i=1}^{m} l(f(x_i, \alpha), y_i) \tag{2.8}$$

where l is the zero-one *loss function*, $l(y, \hat{y})$ if $y \neq \hat{y}$ and 0 otherwise. R_{emp} is called the *empirical risk*, and m represents the *training error* [100].

We are trying to minimize the overall risk [100]:

$$R(\alpha) = \int l(f(x, \alpha), y) dP(x, y) \tag{2.9}$$

where $P(x, y)$ is the (unknown) joint distribution function of x and y. $R(\alpha)$ represents the *test error*.

A second formalization of the process of minimizing misclassification is shown in [79], using the Expected Prediction Error (EPE):

$$\text{EPE} \triangleq \text{E}[L(A, \hat{A}(X))] \tag{2.10}$$

L is a *loss* function, and the expectation E is taken with respect to the joint distribution $P(A, X)$. With $K = |\mathcal{A}|$ classes, the loss function may be represented as a $K \times K$ matrix

L [100]. This loss matrix has values of zero on the diagonal. Everywhere else, a non-zero value $L(k, l)$ indicates the penalty for misclassifying an observation as belonging to $\mathcal{A}_l$ when it actually belonged to $\mathcal{A}_k$ [100].

By conditioning on X, we can rewrite 2.10 as [100]:

$$\text{EPE} = \text{E}_X \sum_{k=1}^{K} L[\mathcal{A}_k, \hat{A}(X)] P(\mathcal{A}_k|X) \tag{2.11}$$

When the loss function is *zero-one*, meaning that a single unit penalty is assessed for any misclassification, the intuitive guidance for $\hat{A}(X)$ is [100]:

$$\hat{A}(X) = \mathcal{A}_k \text{ if } P(\mathcal{A}_k|X = x) = \max_{a \in \mathcal{A}} P(a|X = x) \tag{2.12}$$

In other words, the classification output should be the most probable class given the input for any number of classes. Naturally, what makes this difficult is the fact that one has to estimate the probabilities using a limited set of training data.

In order to provide the best possible classification with limited training data, we review several of the more popular classification techniques [100]. The Support Vector Machine discussed in 2.2.9, which is used in this research effort is considered arguably the best classifier for this type of problem [100]. The basic idea of a SVM is to find the hyperplane that separates the training data with the maximum margin [172]. By maximizing the distance on either side to the nearest samples, the largest buffer possible is created between each class of data decreasing the probability of misclassification. For mathematical details, the reader is referred to [79, 100, 172].

Most methods for classification use numerical values and are unable to handle symbolic information directly. Packet data can contain non-numerical, qualitative data that indicates potential attacks. Converting this data into a numerical structure presents challenges, however it greatly increases the ability for effective classification. Experiments conducted in [80] greatly improved the classifier's accuracy using symbolic conversion.

Outside of Support Vector Machines, other classification algorithms that were considered but not chosen for this research assignment are:

- Gene Expression Programming [138]
- Maximum Entropy Classifier Logistic regression [70]
- Naive Bayes Classifier [148]
- Neural Networks [79]
- Quadratic Discriminant Analysis
- Binary Classifier Tree [100]

Gene expression programming (GEP) is an evolutionary algorithm that creates complex tree structures that learn and adapt by changing their sizes, shapes, and composition, much like a living organism [138]. GEP has been criticized for not being a major improvement over other genetic programming techniques. In many experiments, it did not perform better than existing methods [138].

Maximum Entropy Logistic regression is a regression model which generalizes logistic regression by allowing more than two discrete outcomes [70]. Greene published a book entitled, Econometric Analysis, on this type of classification [70]. Logistic regression assumes that all data is case specific, meaning that no overlap in cases can exist. This along with other assumptions from the text [70] make this classifier incapable of performing the necessary functions for MFIRE.

A Naive Bayes Classifier is a simple approach for the complex classification. A naive Bayes classifier works by assuming the presence of a given feature is unrelated to the presence of any other feature [148]. One advantage of a naive Bayes classifier is that it requires very little training data to estimate the parameters necessary for classification. Because independent variables are assumed, only the variances of the variables for each class need to be determined and not the entire covariance matrix [148]. Comprehensive

comparisons with other approaches, including SVMs and Neural Networks, showed that Bayes classification is outperformed by these and many other approaches [148].

Artificial Neural Networks (ANN) is a wide family of different algorithms and methods. A neural network consists of an interconnected group of artificial neurons, and it processes information using a "connectionist" approach to computation [108]. In most cases, a neural network is an adaptive system that changes its structure during a learning phase [179]. Neural networks are used to model complex relationships between inputs and outputs or to find patterns in data. ANNs are designed much like a decision tree, however through training an adaptive system can move decision nodes to improve classification [25].

Neural Networks were designed to decide between multiple classes, unlike Support Vector Machines which require wrapper methods to convolve the two-class classification to a multi-class classification [108]. For this reason one might consider neural networks the obvious choice for a classifier. Neural Networks are an excellent classifier for many experiments, including intrusion detection, however SVMs have been shown to outperform ANNs in cases with limited training data, good feature selection prior to training, and cases where rapid classification is necessary [179].

Neural networks other advantage over SVMs besides the multi-class design, is that the Neural Network is designed to continually learn (reinforcement learning) while it classifies [108]. This makes Neural Networks valuable for cases where the inputs regularly change. For this reason, Neural Networks are often selected for classifying user interest on product advertisements such as Amazon, Google, and Netflix [177].

2.2.8 Clustering.

Algorithms that derive the decision or discriminant function using prototype patterns or training data are called supervised algorithms for learning [79]. *Clustering* seeks to identify the natural groupings of the data without the use of labeled data and is therefore classified as unsupervised learning. Typically, the number of groupings or *clusters* is not

known beforehand, making the clustering process partially subjective. It is up to the user to determine whether certain groupings are one large cluster, or two, or many based on their relative distances from each other and to the other members of its cluster. The solution is to either simply specify the desired number of clusters and evaluate the resulting cluster assignments, or define some distance-based threshold from which the number of clusters is derived. In a Network Intrusion Detection however, it is more likely to predefine the number of attack classes that exist in an attempt to more accurately cluster the data streams.

Clustering requires a measure of dissimilarity $d_j(x_{ij}, x_{i'j})$ between values of the jth instance [79]. Then

$$D(x_i, x_{i'}) \triangleq \sum_{j=1}^{p} w_j \cdot d_j(x_{ij}, x_{i'j}); \sum_{j=1}^{p} w_j = 1.$$

is the dissimilarity between objects i and i' given the inputs x_i, $x_{i'}$ and weight vector w [79]. Usually, $d_j(x_{ij}, x_{i'j}) = (x_{ij} - x_{i'j})^2$, but other choices are possible, or even required in the case of non-quantitative attributes [79].

Clustering is the search for an encoder $C(i)$ that assigns the ith of N observations to one of K clusters. An encoder may be evaluated by measuring the *between-class scatter* to *within-class scatter* ratio, $\frac{B(C)}{W(C)}$. Between-class scatter is defined as [79]:

$$B(C) \triangleq \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(i')\neq k} d(x_i, x_{i'})$$

while within-class scatter is

$$W(C) \triangleq \frac{1}{2} \sum_{k=1}^{K} \sum_{C(i)=k} \sum_{C(i')=k} d(x_i, x_{i'})$$

We desire high $\frac{B(C)}{W(C)}$ in order to achieve the goal of high between-class scatter and low within-class scatter, thus establishing clusters with well-defined boundaries [79].

A commonly used clustering algorithm is the k-means algorithm, which is based on minimizing a performance index, F [27]. K is the number of clusters specified by the user, and F is the sum of squared distances of all points in a cluster to the cluster center.

The k-means algorithm begins by assigning each observation to the cluster with the closest mean. As stated above, and intuition suggests; there are exactly K means that each observation can be assigned. With each observation assigned to a cluster, the means of each cluster are recalculated with the centroid of each cluster becoming the new mean [27]. With new values for each centroid, the algorithm reassigns each observation to the closest mean, which may have changed due to the shift of the K value [27].

The algorithm is deemed to have converged when the assignments no longer change. In general, there is no guarantee that it converges to the global optimum, and the result is highly dependent on the initial assignments of K. As the k-means algorithm is usually very fast, it is common to run it multiple times with different starting conditions [27].

Other common Clustering Algorithms include: [3]

- Categorical mixture models
- Deep learning methods
- Hierarchical clustering (agglomerative or divisive)
- Kernel principal component analysis

2.2.9 Support Vector Machines.

Support Vector Machine (SVM) were originally used to solve supervised two-class classification problems for use in the field of statistical learning theory [79, 100]. SVMs are now capable of solving one-class and multi-class classification problems and are capable of running in parallel in order to reduce training and classification time [173]. SVMs were developed at AT&T by Vladimir Vapnik and are now a well-known and popular technique for classification and regression [172]. Two-class SVMs solve classification problems by determining an optimal separating hyperplane between the two given classes and are known for relatively fast classification and training despite their high accuracy [100]. SVMs also

are capable of utilizing high-dimensional feature space to optimize the distance between features and the separating hyperplane [173].

SVMs work by a priori learning from observed data. This model of learning by example can be shown as three components:

1. *a* generator of random vectors x, drawn independently from a fixed but unknown P(x);

2. *a* supervisor that returns an output vector y for every input vector x, according to a conditional $P(y|x)$, also fixed but unknown;

3. *a* learning machine capable of implementing a set of functions $f(x, \alpha), \alpha \in \Lambda$; in this case a Support Vector Machine.

The *problem of learning* according to Vapnik, is choosing from the given set of functions $f(x, \alpha), \alpha \in \Lambda$, the one which predicts the supervisor's response most accurately [172]. These selections are made based on the training set of data drawn according to $P(x, y) = P(x)P(y|x)$.

Vapnik described the Problem of Risk Minimization in Statistical Learning Theory and SVMs by showing that in order to choose the best available approximation to the supervisor's response, the learning machine must measure the discrepancy $L(y, f(x, \alpha))$ between the response of y from the given input x and the response provided by the learning machine: $f(x, \alpha)$ [173]. Vapnik explains that the expected value of the loss between the supervisor's response and the learning machine can be shown as a *risk functional* [173]:

$$R(\alpha) = \int L(y, f(x, \alpha))\, dP(x, y) \tag{2.13}$$

The empirical risk of learning machines, specifically SVMs, in using a least square method yields the empirical risk function [173]:

$$R_{emp}(\alpha) = 1/l \sum_{i=1}^{l} (y_i - f(x, \alpha))^2 \tag{2.14}$$

where l is the current feature and $(y_i - f(x, \alpha))^2$ is the mean squared error.

Empirical risk is a quality estimation of the experimental expected performance of the SVM, however the true risk must be accounted for as well in order to illustrate the complexity of the SVM. The true risk can be given by the empirical risk plus an additional term [173]:

$$R(\alpha) \leq R_{emp}(\alpha) + \sqrt{\frac{h(log(\frac{2m}{h} + 1) - log(\frac{n}{4})}{m}} \tag{2.15}$$

where h is the dimensionality of the set of functions parameterized by α. This is a measure of the functions' complexity. The more phenomena that are described, the larger the value of h. Therefore, h is *the maximum number of points that can be separated in all possible ways by that set of functions*.

SVMs operate in vector spaces like many other learning machines. The *dimension* of the vector space is determined by the number of features used. An SVM creates a separating Hyperplane $f(x) = w \cdot \Phi(x) + b$ that separates the individual classes most effectively [172]. In creating the separating hyperplane, w represents the normal vector perpendicular to the hyperplane, b is the offset from the origin, and features are mapped to the higher dimensional space with $x \mapsto \Phi(x)$. As an example the polynomial mapping of a set of features is [172]:

$$\Phi : R^2 \rightarrow R^3 (x_1, x_2 \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{(2)x_1 x_2}, x_2^2) \tag{2.16}$$

Figure 2.5 illustrates two hyperplanes separating the same data in sections (a) and (b). The hyperplane in part (a) of the Figure does not maximize the margin to the two surrounding lines. This hyperplane increases the chance that a faulty classification because there is less of a margin of error between the two classes. The hyperplane of part (b) of Figure 2.5 separates both data sets optimally. The margin of part (b) to the two surrounding

lines, representing the class borders, is maximized. The classification of a vector is performed by determining on which side of the hyperplane the vector lies to determine which class it belongs. By increasing the margin it becomes easier to determine which side of the hyperplane the vector belongs on, thus increasing the classification accuracy. The SVM in particular defines the criterion to be looking for a decision surface that is maximally far away from any data point. This distance from the decision surface to the closest data point determines the margin of the classifier. This method of construction means that the decision function for an SVM is fully specified by a subset of the data which defines the position of the separator. *These points are referred to as the support vectors*.

Figure 2.5 shows the margin and *support vectors* for a sample problem. Other data points play no part in determining the decision surface that is chosen [120].

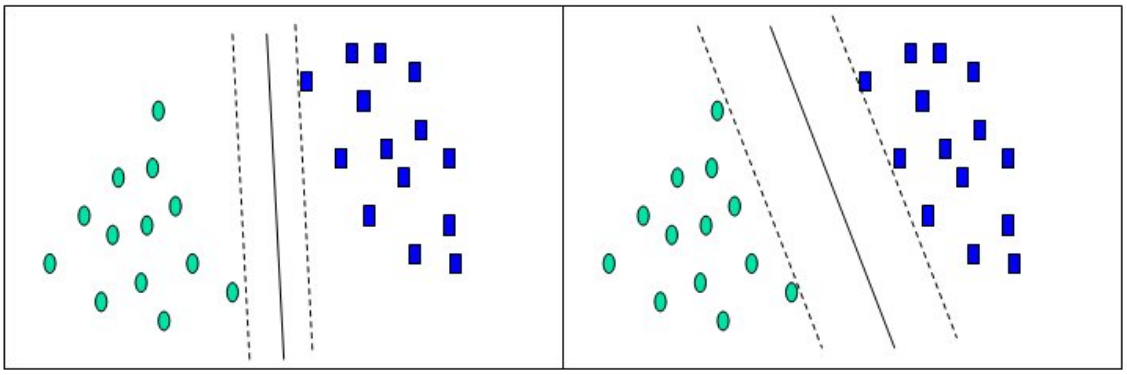

Figure 2.5: Poor (a) and Optimal (b) separating hyperplanes of an SVM. The poorly separating hyperplane offers bad generalization ability whereas the optimal separating hyperplane perfectly divides both data sets by maximizing the margin of the hyperplane [120]

As stated above, Part (a) of Figure 2.5 features a non-optimal hyperplane. The margin of the hyperplane is visibly smaller than the margin in part (b). How this effects

classification accuracy is in the generalization ability since vectors lying very close to the hyperplane can be misclassified more easily. *The difficulty of training an SVM now lies in finding the "optimal" separating hyperplane*. The hyperplane is calculated from a training set $D = (\vec{x_i}, y_i)$, where D is the data points used in the training of the classifier, and each member is a pair of points $\vec{x_i}$ and a class label, y_i corresponding to it. A decision hyperplane is defined by an intercept term b and a decision hyperplane normal vector $\vec{\omega}$ which is perpendicular to the hyperplane also called the *weight vector* [100]. To choose the "best" hyperplane that is perpendicular to the normal vector, use intercept term b. All points $\vec{x_i}$ on the hyperplane satisfy $\vec{\omega}^T \vec{x} = -b$. The linear classifier then becomes [173]:

$$f(\vec{x}) = sign(\vec{\omega}^T \vec{x} + b) \tag{2.17}$$

And the value of -1 indicates not in the current class, where a value of 1 indicates in the current specified class. By moving the hyperplane to a location *farthest away* from points of the two classes, the *optimal* hyperplane is reached.

The dimensionality and complexity of $\Phi(x)$ can be very large ($O(n^n)$), making w hard to represent in memory, and hard to solve [172]. Kimeldorf and Wahba [96] presented the representer theorem, which shows that

$$w = \sum_{i=1}^{m} \alpha_i \Phi(x_i) \tag{2.18}$$

for some variables α. Instead of optimizing w directly we can optimize α [96]:

$$f(x) = \sum_{i=1}^{m} \alpha_i \Phi(x_i) \cdot \Phi(x) + b \tag{2.19}$$

and $K(x_i, x) = \Phi(x_i) \cdot \Phi(x)$ is called the *kernel function*.

So far only the linear classification capability of SVMs is introduced. Nonlinear classification by means of kernel functions are able to separate data which might not seem linearly separable in an SVM. SVMs are also cpable of classifying prefiltered data [120].

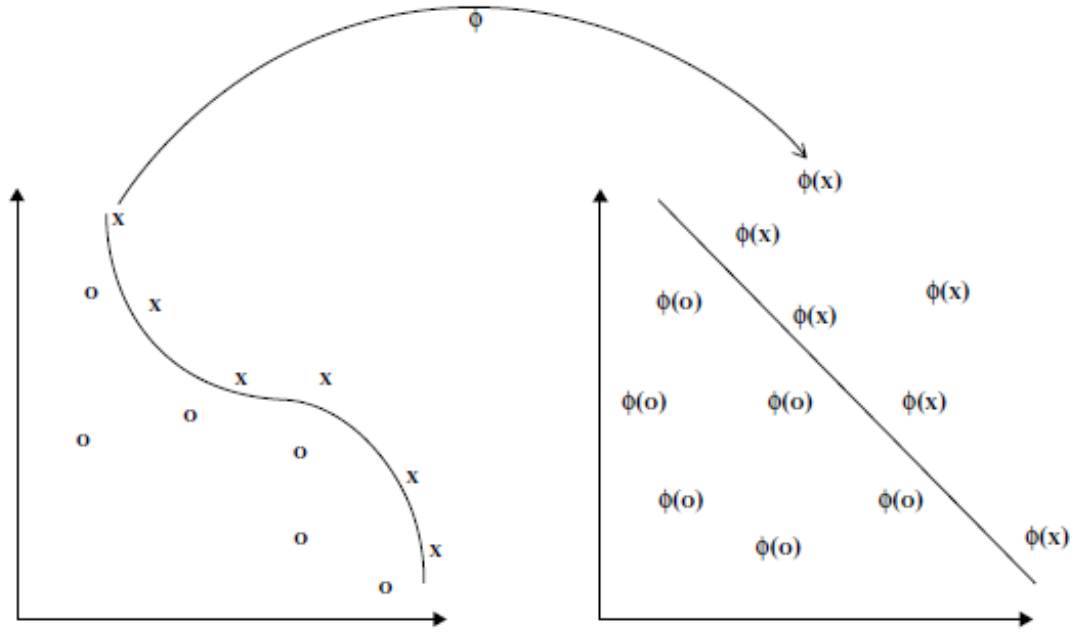

Figure 2.6: SVM separating features with a hyperplane in a higher dimensional space [96]

Figure 2.6 illustrates a data set that would not be linearly separable without being brought into a higher dimensional space. The two data sets are not linearly separable without accepting many training errors because many points would reside on the wrong side of the hyperplane. To solve the nonlinear classification problem, kernel functions, defined as $K(X, Y) = [\rho(x), \rho(y)]v$, are used [96]. The purpose of *the Kernel Trick is to transform vectors from the lower dimensional input space to the higher dimensional feature space in which the data sets become linearly separable* [172]. The Kernel often uses the Gaussian radial basis function kernel, $k(x_i, x_j) = exp(-\lambda||x_i - x_j||^2)\ for\ \lambda = 1/2\phi$ [100]. Other Kernels for nonlinear classification include Polynomial homogeneous, Polynomial inhomogeneous, and Hyperbolic Tangent [118].

SVMs have shown good results in data classification, but their training complexity is very dependent on the size of the dataset. SVMs are known to be at least *quadratic* ($O(n^4)$) with the number of training data points. One approach to reduce training data size is to

use a hierarchical clustering algorithm, as described by Horng [85]. The algorithm creates a clustering feature tree, which is then used to merge disjoint clusters. Experiments using this technique on the intrusion detection problem are encouraging [85].

Another interesting use of SVMs in the intrusion detection problem is introduced in [166], which evaluates a hybrid decision-tree/SVM system. Their hypothesis is that different classifiers are better at detecting certain attacks than others, and that an ensemble approach using several different classifiers can exploit the misclassification and improve performance. Experimental results support this assessment [166].

To this point, the only discussion of SVMs is two class binary problem solvers. Multiclass SVMs are able to assign labels to instances for a finite set of classes. The dominant method for solving a multiclass SVM is to reduce the multiclass problem into several binary classification problems and combining the results [101].

The common methods for reducing multiclass decision problems are one of the labels and the res, known as the *one-versus-all* approach, or between every pair of classes; *one-versus-one*. Other methods for such reduction include the use of Directed Acrylic Graph SVM (DAGSVM) and error correcting output code [101].

Crammer and Singer [157] propose the use of an SVM method which casts the multiclass classification problem into a single optimization problem. This avoids the process of decomposing the problem into multiple binary classification problems.

Crammer and Singer propose that starting with set $S = (\vec{x}_1, y_1, ..., \vec{x}_m, y_m)$ be a set of m training samples, a multiclass classifier maps using the function $H : X \rightarrow Y$ each instance of $\vec{x}$ to an element y of Y. The form of the classification process is [157]:

$$H_m(\vec{x}) = arg\ max_{r=1}^{k}[\vec{M_R} \cdot \vec{x}] \tag{2.20}$$

where $\boldsymbol{M}$ is a matrix of size kn over $\boldsymbol{R}$ and $\vec{M}_r$ is the r^{th} row of $\boldsymbol{M}$. The inner product of the r^{th} row of M is the similarity score or confidence of the r class. To construct a multiclass predictor, the misclassification error becomes the following piecewise linear bound [157]:

$$max_r[\vec{M}_r \cdot \vec{x} + 1 - \delta_{y,\,r}] - \vec{M}_y \cdot \vec{x}, \tag{2.21}$$

where $\delta_{p,q}$ equals 1 if p = q and 0 otherwise. The above bound is zero if the confidence value for the correct label is larger by at least one than the confidences assigned to the rest of the labels. Otherwise, the value suffers a loss which is linearly proportional to the difference between the confidence of the correct label and the maximum among the confidences of the other labels. A graphical illustration of the above is given in Figure 2.7. The circles in the figure denote different labels and the correct label is plotted in dark grey while the rest of the labels are plotted in light gray. The height of each label designates its confidence. Three settings are plotted in the figure. The left plot corresponds to the case when the margin is larger than one, and therefore the bound $max_r[\vec{M}_r \cdot \vec{x} + 1 - \delta_{y,\,r}] - \vec{M}_y \cdot \vec{x}$ equals zero and is correctly classified. The middle figure shows a case where the example is correctly classified but with a small margin and suffers some loss. The right plot depicts the loss of a misclassified example.

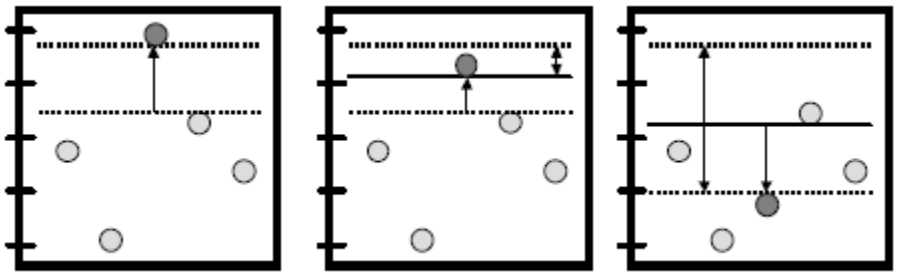

Figure 2.7: Margin Bound for Multiclass SVM [185]

Although SVMs apply optimal classification techniques when binary classification is necessary, when decomposed into multiple binary classification problems this is not always the case for multiclass problems [109]. [110] shows an alternative for multiclass problems using a Multicategory SVM. The multicategory SVM implements the "optimal" classification rule as the sample size gets large, overcoming the sub-optimality of the conventional one-versus-rest approach. Their method deals with the equal misclassification cost and the unequal cost case in unified way.

The basic principals of the method are similar to those of Crammer and Singer, where a matrix is formed of the j possible classes, and a data point belonging to that class is mapped to that class with a value of 1. If it is not part of that class it is marked with a -1. The main problem with this then becomes over-fitting the data. Misclassification leads to a penalty if the data is not separable, however fitting the data too closely leads to a penalty as well. This set of rules allow the SVM using Gaussian Radial Basis function to use the best dimension in finding all of the linear hyperplanes. The crux of their methodology is the following Lemma:

Lemma: The minimizer of $E[L(Y) \cdot (f(X) - Y)_+]$ under the sum-to-zero constraint is: $f(x) = (f_1(x), ..., f_k(x))$ with: $f_j(x) = \{1 \; if \; j = arg \; max_{l=1,...,k} p_l(x), \;\; -1/k \; otherwise\}$.

Notice that the minimizer is exactly the representation of the most probable class. Hence, the classification rule induced by $f(x)$ is naturally ϕ(x) = arg max_j f_j(x). If $f(x)$ is the minimizer in the Lemma, then the corresponding classification rule is $\phi - B(x) = arg \; max_j \; p_j(x)$, the Bayes rule for the standard multicategory case.

Support Vector Machines suffer from a scalability problem in both memory use and computational time. To improve scalability developers have created a parallel SVM algorithm (PSVM) [109], which reduces memory use through performing a row-based, approximate matrix factorization, and which loads only essential data to each machine to perform parallel computation. Let n denote the number of training instances, p the reduced

matrix dimension after factorization, and m the number of machines or cores. PSVM reduces the memory requirement from $O(n^2)$ to $O(np/m)$, and improves computation time to $O(np^2/m)$.

A short subset of popular SVM packages is located in Appendix D.

2.3 Intrusion Detection

The previous section described the field of pattern recognition, including classification. Applying classification techniques on the AS-level of a network allows for the detection and characterization of malicious activity.

Intrusion Detection Systems (IDS) fall into two pairs of categories: host-based or network-based; and anomaly-based or signature-based 9. Recognizing malicious activity on a computer network is called Intrusion Detection, and it is the job of the IDS [29].

A Host-Based Intrusion Detection System (HIDS) monitors and analyzes the network packets on its network interfaces. It was the first type of IDS and consists of an agent on a host that identifies intrusions by analyzing system calls, application logs, file-system modifications, and other host activities. A HIDS monitors the dynamic behavior and the state of the network. In a HIDS, sensors usually consist of a software agent. An example of a HIDS is OSSEC [42].

A Network Intrusion Detection System (NIDS) is an independent software platform that identifies intrusions by examining network traffic and monitors multiple hosts. Its main function is to discover unauthorized access to a computer network by analyzing traffic on the network for signs of malicious activity. Network intrusion detection systems gain access to network traffic by connecting to a network hub, network switch configured for port mirroring, or network tap. In a NIDS, sensors are located at choke points in the network to be monitored. Sensors capture all or part of network traffic and analyze the content of individual packets for malicious traffic. An example of a NIDS is Snort, an open source NIDS developed by Martin Roesch and maintained by Sourcefire Inc. [1].

An Anomaly-Based Intrusion Detection System (ADS) works by detecting computer intrusions and misuse by monitoring system activity and classifying it as either normal or anomalous. Typically, these systems begin by determining normal operating conditions for bandwidth, protocols, ports and device connections. The classification is based on heuristics or rules, rather than patterns or signatures, and detects any type of misuse that falls out of normal system operation. This opposes signature based systems which can only detect attacks for which a signature has previously been created [9].

In order to determine what is attack traffic, the system must be taught to recognize normal system activity. This is most often accomplished with artificial intelligence techniques, including neural networks and classifier systems. Another method, known as strict anomaly detection, is to first define the normal usage of the system using a strict mathematical model, and flag any deviation from this as an attack. CFEngine developed by Mark Burgess has support for this technique [33], as well as RRDTool by Tobi Oetiker [137].

Anomaly-based Intrusion Detection does have some short-comings, namely a high false positive rate and the ability to be fooled by a correctly delivered attack. Attempts have been made to address these issues through payload-based techniques used by PAYL [178] and MCPAD [142]. Signature-based systems have a very low false-positive rate, but are more limited in the types of attacks they can detect. Novel attacks which are designed to thwart signature-based systems may still be detectable by an anomaly-based system.

Details of terminology and specific concepts of intrusion detection are further discussed in Appendix E.

2.4 Flow-based Intrusion Detection

The traditional idea of a network *flow*, as defined in [187], is a unidirectional data stream between two computer systems where all transmitted packets of this stream share the following characteristics: IP source and destination address, source and destination

port, and IP protocol. Thus, all network packets sent from host A to host B sharing the above mentioned characteristics form a flow. Every communication attempt between two computer systems triggers the creation of a flow, even if no connection is established. In the simplest case, a complete flow is well-defined when a complete flow set-up and tear-down are observed, as is the case with most TCP communications. Complexity in any flow definition occurs when the set-up is incomplete or tear-down is abnormal. UDP is notoriously troublesome because it is connectionless protocol.

In addition to the above mentioned core characteristics, several other properties of a flow can be conveyed, for instance:

- The number of packets which have been transferred
- The number of bytes which have been transferred
- The start or end time of a flow
- The disjunction of all TCP flags occurring in the flow

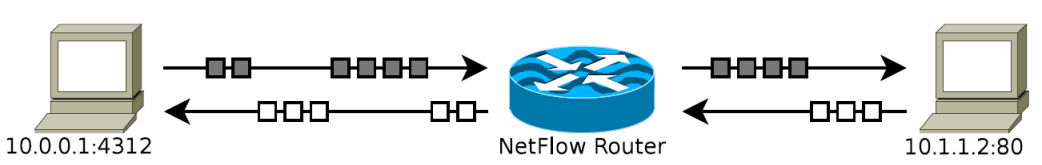

Figure 2.8: Network flow

Figure 2.8 illustrates a bidirectional communication between two computers which results in the creation of two flows. Host A is the initiator of the communication and has the IP address 10.0.0.1. Host A sent several packets to host B which is assigned the IP address 10.1.1.2. The source port of this communication is 4312 on host A whereas the destination port is 80 on host B. All the network traffic is monitored by the NetFlow

router. The communication finally results in two unidirectional network flows. The first flow (illustrated as grey squares) describes the communication from A to B and the second flow (illustrated as white squares) from B to A.

Winter [187] describes a technique to collect network flows on actual hardware, with a commercial package called NetFlow. NetFlow runs on Cisco routers and collects flow statistics which it sends to a central collector. A separate device can poll this collector to run analysis on current flows in the network. MFIREv3 does not use live network flows – instead all traffic is simulated. However the NetFlow architecture provides a well-known framework for modeling flows, and this model is useful in discussing this research investigation.

A useful set of real-world flow data and metrics is provided by Andrew Moore [133]. Real network traffic was collected over a 24-hour period at a research facility with approximately 1000 active workstations. Individual flows are constructed from this data, and labeled as *idle*, *interactive* (two-way), or *bulk* (one-way). Only data and metrics corresponding to the TCP protocol are collected; UDP and ICMP are ignored. Flows are characterized into 249 metrics.

However, one does not need to observe a specific TCP connection or tear-down to use flows. A *microflow* abandons such concepts in favor of observing traffic in a more immediate fashion. This concept treats flows as a collection of packets to/from nodes, but does not distinguish bi-directional flows; everything is treated as one-directional. These flows are robust to incorrectly formatted TCP connections and tear-downs because they do not rely on those actions for measurement. A disadvantage is that microflows lose potentially useful information, including the cumulative time that a connection has been established, or the amount of data sent since the beginning of a connection. A good comparison between the usefulness of both approaches for the ID problem is provided in [177].

In the environment used in this research effort, TCP is not specifically implemented; rather everything behaves like UDP. Because of this, mircroflows are the obvious choice.

2.5 Network Attacks of Interest

This section discusses five common types of network attacks: distributed denial of service, vulnerability scans, worm propagation, man-in-the-middle, and Trojans or Trojan Horse attacks. A taxonomy of network and computer attacks is located in Appendix B. We focus on attacks which cause significant changes in traffic flows. Background on other attacks can be found in [159].

2.5.1 Denial of Service Attacks.

Mirkovic [130] presents a comprehensive taxonomy of different DDoS attack types, Figure 2.9. We concentrate on flood attacks [162], although the MFIREv3 environment is capable of simulating a Semantic or other DoS model as well. A flood attack involves malicious agents sending large volumes of traffic to a victim system, to congest the victim system's network bandwidth with IP traffic *with the victim's own legitimate resources*. The victim system slows down, crashes, or suffers from saturated network bandwidth, preventing access by legitimate users.

Formal models for DDoS and their detection are proposed in the several articles. One method applied to DDoS detection is the k-nearest neighbor (kNN) algorithm with feature weighting and selection based on a genetic algorithm [129]. Overall accuracy of over 97% for known DDoS attacks is achieved, and over 78% in the case of unknown attacks.

Scepanovic [150] focuses on the scenario in which a cluster-based filter is deployed at the target network and serves for proactive or reactive defense. A game-theoretic model is created for the scenario, making it possible to model the defender and attacker strategies as mathematical optimization tasks. The model is based on the continuous nonlinear knapsack

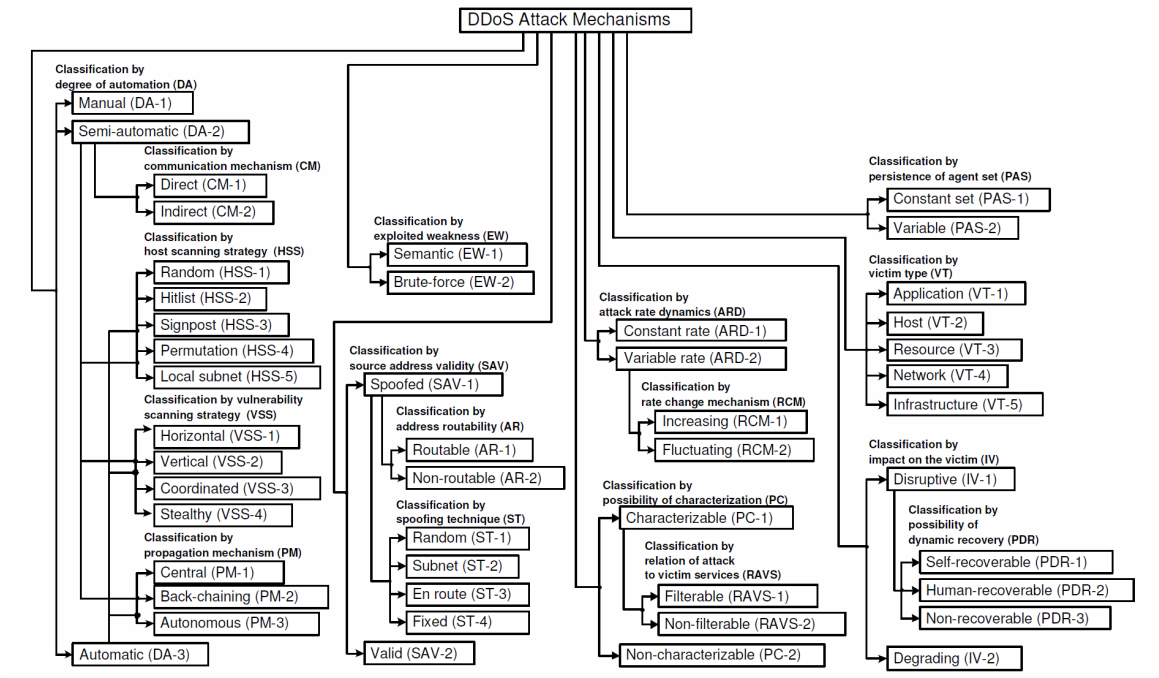

Figure 2.9: Taxonomy of DDoS Attack Mechanisms [122]

problem [66]. The experimental outcome shows the high effectiveness of cluster-based filtering in proactive and reactive DDoS defense.

Once service is denied or begins suffering bandwidth reduction the DDoS is detected, however early detection is important [141]. If a target can detect an attack before the actual damage occurs, the target can win more time to implement attack reaction and protect legitimate users. Second, if attacks can be detected close to attack sources, attack traffic can be filtered before it wastes any network bandwidth. However, there is generally insufficient attack traffic in the early stage of an attack and at links close to attack sources. Consequently, it is easy to mistake legitimate traffic as attack traffic. Therefore, it is challenging to accurately detect attacks quickly and close to attack sources.

Finally, *flash crowds* are very similar to DoS attacks, which can cause network congestion and service degradation. However, flash crowds are caused by legitimate traffic,

whereas DoS attacks caused by malicious traffic. Hence, it is important to differentiate DoS attacks from flash crowds so that targets can react to them separately.

DoS attacks can be easily detected since the target's services degrade as the attack manifests. False positives are a serious concern for DoS attack detection. Since the potency of DoS attacks does not depend on the exploitation of software bugs or protocol vulnerabilities, it only depends on the volume of attack traffic. Consequently, DoS attack packets do not need to be malformed, such as invalid fragmentation field or malicious packet payload, to be effective [141]. As a result, the DoS attack traffic can look very similar to legitimate traffic.

2.5.2 Vulnerability Scans.

A vulnerability scan is used to conduct network reconnaissance. A remote attacker usually attempts to gain information or access to a network on which it is not authorized or allowed. Network reconnaissance is increasingly used to exploit network standards and automated communication methods. The aim is to determine what types of computers are present, along with additional information about those computers; such as the type and version of the operating system. This information can be analyzed for known or recently discovered vulnerabilities that can be exploited to gain access to secure networks and computers. Network reconnaissance is possibly one of the most common applications of passive data analysis. Numerous tools exist to make reconnaissance easier and more effective.

A port scan is an attack that sends client requests to a range of server port addresses on a host, with the goal of finding an active port and exploiting a known vulnerability of that service [154]. The result of a scan on a port is usually generalized into one of three categories:

- *Open*: The host sent a reply indicating that a service is listening on the port.
- *Closed*: The host sent a reply indicating that connections are denied to the port.

- *Filtered*: There was no reply from the host.

Potential security concerns exist for both the program responsible for delivering a service (on open ports), and with the operating system that is running on the host (on open or closed ports). Filtered ports do not tend to present vulnerabilities. There are many standard scanning formats, some of which follow standard Internet protocols, others which purposefully do not [159]. Some common techniques are outlined:

TCP CONNECT scan—The simplest port scanners use the operating system's network functions. If a port is open, the operating system completes the TCP three-way handshake, and the port scanner immediately closes the connection. Otherwise an error code is returned. This scan mode has the advantage that the user does not require special privileges. However, using the OS network functions prevents low-level control, so this scan type is less common. This method is noisy, particularly if it is a complete sweep of all ports: the services can log the sender IP address and Intrusion detection systems can raise an alarm.

TCP SYN scan—SYN scan is another form of TCP scanning. Rather than use the operating system's network functions, the port scanner generates raw IP packets itself, and monitors for responses. This scan type is also known as "half-open scanning", because it never actually opens a full TCP connection. The port scanner generates a SYN packet. If the target port is open, it responds with a SYN-ACK packet. The scanner host responds with a RST packet, closing the connection before the handshake is completed.

The use of raw networking has several advantages, giving the scanner full control of the packets sent and the timeout for responses, and allowing detailed reporting of the responses. SYN scan has the advantage that the individual services never actually receive a connection. However, the RST during the handshake can cause problems for some network stacks, in particular simple devices like printers.

UDP scan—UDP is a connectionless protocol so there is no equivalent to a TCP SYN packet. However, if a UDP packet is sent to a port that is not open, the system responds with

an ICMP port unreachable message. Most UDP port scanners use this scanning method, and use the absence of a response to infer that a port is open. However, if a port is blocked by a firewall, this method falsely reports that the port is open. If the port unreachable message is blocked, all ports appear open. This method is also affected by ICMP rate limiting.

An alternative approach is to send application-specific UDP packets, hoping to generate an application layer response. For example, sending a DNS query to port 53 results in a response, if a DNS server is present. This method is much more reliable at identifying open ports. However, it is limited to scanning ports for which an application specific probe packet is available. Some tools (e.g., nmap) generally have probes for less than 20 UDP services, while some commercial tools (e.g., nessus) have as many as 70. In some cases, a service may be listening on the port, but configured not to respond to the particular probe packet.

TCP ACK scan—ACK scanning is one of the more unique scan types, as it does not exactly determine whether the port is open or closed, but whether the port is filtered or unfiltered. This is especially good when attempting to probe for the existence of a firewall and its rulesets. Simple packet filtering allows established connections (packets with the ACK bit set), whereas a more sophisticated stateful firewall might not.

TCP FIN scan—Firewalls are, in general, scanning for and blocking covert scans in the form of SYN packets. FIN packets are able to pass by firewalls with no modification to its purpose. Closed ports reply to a FIN packet with the appropriate RST packet, whereas open ports ignore the packet on hand. This is typical behavior due to the nature of TCP, and is in some ways an inescapable downfall.

2.5.3 Worms.

It is vital to detect active worms effectively. In the near future active worms may spread across the whole Internet in a very short period of time, making the average detection time

critical. A common way to detect worms is to place sensors in a network to monitor messages sent to non-existent IP addresses. Administrators of networks are aware of exactly which IP addresses are in use in their domains, and common worm attacks do not have access to this information. If a message is sent to a non-existent IP, then this flags the sender as suspicious [39]. Attackers that wish to build stealth into the system must take preliminary steps to discover a network map prior to initiating the worm.

Many models exist for worm propagation [39, 95, 111, 151, 176, 193, 194]. The basis of many of these is the *general epidemic* model, which considers a fixed population size N where each individual can be in one of three states: susceptible to the disease (S), infected (I), or removed (R) [111]. In networking terms, removals can occur if the victim is taken offline or becomes immune (patched) to the infection. The normal state progression for an individual is $S \rightarrow I \rightarrow R$, normally termed an SIR model. But in the networking domain, victims who recover and do not obtain immunity to the infection become susceptible again: $S \rightarrow I \rightarrow S$, an SIS model. Also known as the *Epidemiological Model*, this is formally represented as:

$$\frac{dn}{dt} = \beta(1-n) - dn \tag{2.22}$$

where $n(t)$ is the fraction of infected nodes, β is the infection parameter, and d is the death rate. The solution to the above equation is

$$n(t) = \frac{n_0(1-\rho)}{n_0 + (1-\rho-n_0)e^{-(\beta-d)t}} \tag{2.23}$$

where $\rho = \frac{d}{\beta}$ and $n_0 \equiv n(t=0) = \frac{sizeofhitlist}{N} = \frac{h}{N}$

2.5.4 Man-in-the-Middle and Eavesdropping.

The Man-in-the-Middle (MiM) attack is a form of active eavesdropping in which the attacker makes independent connections with the victims and relays messages between them, making them believe that they are talking directly to each other over a private connection, when in fact the entire conversation is controlled by the attacker [154]. The

Table 2.1: Parameters for the spread of active worms [111]

vulnerable machines	N	number of vulnerable machines
Size of hitlist	h	number of infected machines at the beginning of the spread of active worms
Scanning rate	s	average number of machines scanned by an infected machine per unit time
Death rate	d	rate at which an infection is detected on a machine and eliminated without patching
Patching rate	p	rate at which an infected or vulnerable machine becomes invulnerable

attacker must be able to intercept all messages going between the two victims and inject new ones, which is often a simple task for an attacker [111].

A MitM attack can succeed only when the attacker can impersonate each endpoint to the satisfaction of the other often times requiring the issuing of false authentication. Most cryptographic protocols include some form of endpoint authentication specifically to prevent MitM attacks. For example, SSL can authenticate one or both parties using a mutually trusted certification authority.

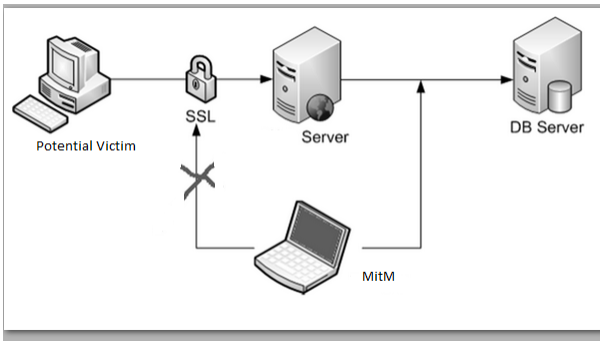

Figure 2.10: Illustration of SSL Thwarting MitM Attack

MitM attacks do not normally flood networks with traffic or attempt to spread once they reach the target node of a system. This makes detection unique for the MFIRE system, as there is very little indication that the flow between two or more nodes has been changed. However, compared to normal traffic flow the packet exchange rate becomes statistically noticeably slower [169]. This concept is illustrated in an effort by Tartakovsky et al. in 2.11.

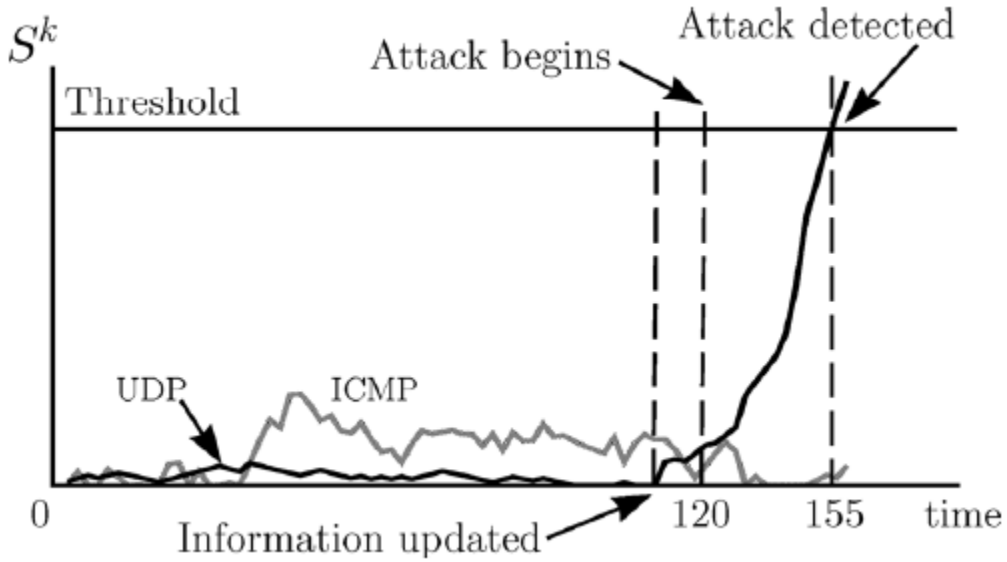

Figure 2.11: Detection of MitM Attack in UDP Mode [169]

According to cyber security experts, "the man-in-the-middle attack has been shown to be one of the most serious threats to the security and trust of existing protocols and systems" [192]. A major source of the threat is the inability to permanently remove the threat from the system and detection is difficult if the attacker remains quiet.

Quite simply, MitM attacks are very difficult to perform on systems with SSL or similar authentication. Advanced attacks though might be capable of deceiving the authentication process. The speed of dataflow between nodes decreases by definition with

a MitM and detection of this slowdown is essential. On a broad spectrum, the detection of MitM due to decreased flow speeds is shown as a good detection mechanism by many research attempts [169]. Without propagation throughout the network however, no tests have proposed solutions for detecting the attack on a large network with few intrusion detection agents. Where the detection of a worm, DDoS, or scan becomes an issue of catching the intrusion before it propagates too far, MitM presents the issue of having the attack located at all on a very large system.

2.5.5 Trojans and Back-doors.

Unlike the worm, scan, and DDoS, a Trojan Horse is utilized for the creation of a vulnerability. A Trojan is a malicious application that masquerades as a legitimate file but instead enables an attacker unauthorized access to the system. Trojans do not attempt to inject themselves into other files like a computer virus or worm, but rather create a back door, destroy a single system, or steal information by updating it to an outside source. Trojans may use drive-by downloads or install via internet-driven applications in order to reach target computers. Trojans often utilize *social-engineering* in order to install themselves on the host-computer [131].

Rather than focusing on the Trojan itself, the detection goal is truly locating the back-door created by the initial attack. Creating a back door allows unauthorized access to the system for an attacker from which the attacker could easily run any number of other attacks such as DDoS, worms, and scans, or simply steal valuable information. As long as the back-door remains ajar, the system remains highly vulnerable even if initial attacks are thwarted [131]. It is even possible that the initial attacker leaves a back-door open without intent to attack the system again, and a new attacker, finding the vulnerability via network scan, exploits the vulnerability for painless access to the system.

Back-doors are typically very difficult to detect if they are not known by the standard anti-virus software [131, 139]. Trojans conceal their presence by executing as a plug-in

or as a dynamically-linked library. Many have very little immediate impact on the normal operation of a system and so are difficult to detect by the user. These characteristics enable Trojan Horses and Back-doors to go undetected for a significant period of time, providing the attacker with a large window of vulnerability on the system [131, 139].

A small subset of the known Trojan exploits include [139]:

- PWSteal.Tarno.Q logs passwords and information typed into specific webforms such as banks. Propagating via email attachments, this Trojan registers as a browser helper similar to a toolbar. The stolen information is periodically transmitted back to the attacker via hard-coded url.

- Trojan.Lodeight.A code tries to install malicious code on the compromised computer and opens a Back-door on TCP port 1084. When this Trojan is executed, it connects to one of two predefined websites, downloads a remote file and then executes it. This remote file may be any arbitrary program, including a Beagle worm.

- Trojan.Vundo is part of an adware program that presents the user with pop-up advertisements. By exploiting a Microsoft Internet vulnerability, a downloader component is executed on the victim. It then retrieves an adware component by connecting to a specific IP address. The adware is injected into dierent processes as a DLL. Besides displaying advertisements on the infected machine, it also degrades performance by decreasing the amount of virtual memory available

Trojans and Back-doors display five common characteristics [146]:

- The malicious code is executed without user intervention.

- The malicious code may be directed by a remote attacker once a connection is made.

- Resources used by the malicious code, such as file names and network addresses, are hard-coded in the binary.

- OS resources (processes, memory) used by the malicious code may be consumed for the purpose of degrading performance.
- They cannot be invoked by the attacker and are autonomous at least until a connection is made.

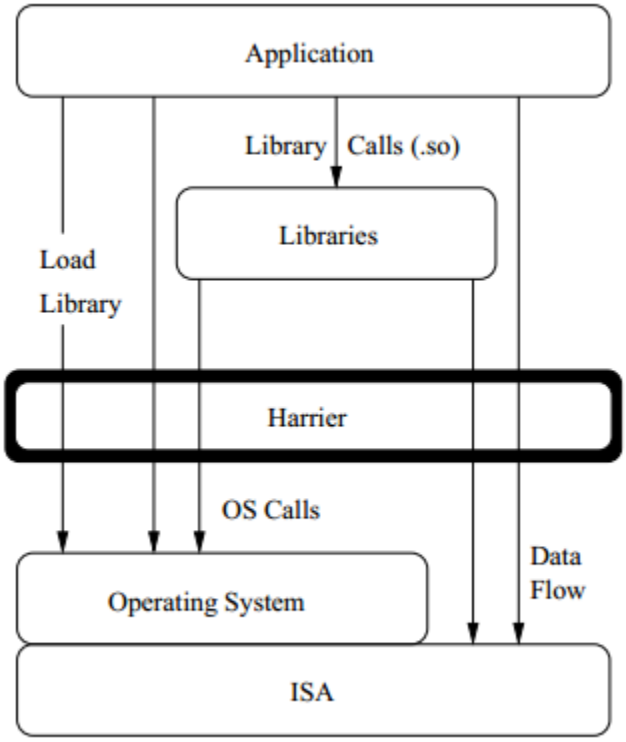

Figure 2.12: Event Classification Abstraction Levels for Harrier [40]

One formal model of the attack and detection of a back-door intrusion by Cheng et al. [40] called Harrier uses the approach of breaking data flows into five categories of resource types: User Input, File, Socket, Binary, and Hardware. Besides monitoring the source of the data flow, which is important for detecting back-door access to the Internet as opposed

to a legitimate user, their model also incorporates Event Monitoring illustrated in Figure 2.12.

2.6 Multi-Objective Optimization

A Multi-Objective Evolutionary Algorithm (MOEA) is utilized for finding the "optimal" subset of a set of points representing conflicting objectives. Multi-Objective Optimization (MOO) is the process of simultaneously optimizing two or more conflicting objectives under certain constraints, and a useful process for this optimization is MOEAs. Multi-Objective Optimization is also known as Multi-Objective Programming, Pareto Optimization, Multi-Criteria Optimization, or Multi-Attribute Optimization [103]. Pareto Optimization refers to the process of finding the optimal or Pareto front of points for all of the conflicting criteria. Since two or more conflicting objectives require multiple solutions where one objective improves to the detriment of the conflicting objectives, a front of all of the optimal solutions is produced. This front of non-dominated, optimal solutions is known as the Pareto Front [103].

2.6.1 Multi-Objective Problems.

Multi-Objective Problems (MOPs) exist in almost every field of business and decision processes. Common scenarios requiring optimization over multiple criteria include any business decision with cost *vs.* quality. Optimal decisions need to be taken in the presence of trade-offs between two or more conflicting objectives [63].

For nontrivial MOPs, one solution that optimizes every objective does not exist. While searching for solutions, improving one objective further often results in the decline of the other objectives [63]. A tentative solution is called non-dominated or Pareto optimal if it cannot be eliminated from consideration by replacing it with another solution which improves an objective without worsening another one. Finding such non-dominated solutions leads to the development of an optimal front of solutions where choosing any solution from the front is not any worse than another solution, because any other choice

would require at least one objective to suffer for the improvement of other objectives [168]. After a Pareto Front is created the decision maker must choose what solution from the front is best based on what the decision maker determines is most important to keep and what is an acceptable sacrifice when it comes to the multiple objectives [63, 103].

2.6.2 Stochastic Search.

In a complex Network Intrusion Detection System agents take on a number of parameters determining attack classification, agent movement, sensors, and any other parameter improving the system. With the large degree of data gathered by each parameter and the complexity of meeting multiple objectives, finding the optimum solution or solutions is not possible. In many cases, one or more of the objectives is a Non-Polynomial Complete (NPC) problem. Stochastic search techniques allow the location of good solutions quickly that are not necessarily optimum solutions.

Recognizing a good solution for a Multi-Agent System or other MOP is an easier function to perform than finding a good solution. Recognizing a good solution simply requires comparing the current solution to the previous solutions or a set of previous options, while finding a good solution requires searching intractable space for optimal solutions. Stochastic search algorithms allow for finding near-optimal solutions to problems with large numbers of solutions along a Pareto Front [168]. Stochastic search is useful on problems that can be formulated as finding a solution maximizing or minimizing a criterion among a number of candidate solutions. Search algorithms move from solution to solution in the space of candidate solutions by applying local, until a solution deemed optimal is found or a time bound is elapsed.

The hill climbing local search algorithm is a commonly known, simple local search algorithm. Hill climbing involves starting with an arbitrary or previously chosen solution, followed by attempting to incrementally find a better solution by changing a single element of the solution utilizing Depth-First Search (DFS). If the change produces a better solution,

that change becomes the new solution. This process repeats until the final solution cannot make any incremental changes that improve its state [11].

A problem with simple hill climbing is that it suffers from a tendency to get stuck at a local optimum and fails to search for greater global optima. Some improvements to the algorithm attempt to mitigate this tendency and allow the search to move to global optima include stochastic hill climbing, random-restart hill climbing, hill climbing with backtracking and Tabu search [43].

An Evolutionary Algorithm (EA) is a type of stochastic search that utilizes the principals of biological evolution as inspiration for local search. Reproduction, mutation, recombination, and selection applied to candidate solutions or individuals of the population allow for the algorithm to evolve towards optimal solutions [11, 43, 168]. EAs evolve solutions over generations of applying the organic operator methods. The evolutionary process aims to keep a balance of the best candidates as well as a number of non-optimal solutions in order to keep diversity in future generations and avoid moving towards a local optima.

Bäck Evolutionary Algorithm Formalism [11]: Optimization as a minimization of a function $f : M \subseteq \mathbb{R}^n \to \mathbb{R}$, $M \neq \emptyset$ consists of searching for $\vec{x}^* \in M$ such that $f(\vec{x}^*) > -\infty$ and

$$\forall \vec{x} \in M : f(\vec{x}^*) \leq f(\vec{x})$$

The goal is to find the global optimum $\vec{x}^*$ for the objective function f within the feasible region M, however time constraints make reaching this goal unlikely. Instead, the best solution found given time or another constraint is selected.

Evolutionary computation begins by initializing a *population* of candidate solutions. Each candidate solution consists of a vector of parameters proposed as solutions for the optimization problem. The *fitness* of candidate solutions is then measured by the

given parameters and candidate solutions are then ranked or graphed for selection in the reproduction phase of the algorithm. In the case of a single objective EA, the quality of a solution could be measured as a single value. For a MOP the strength of a solution could still be measured as a single value if it is simply a weighted sum of all of the objectives, however it is more common for the quality to be represented as a vector. Members chosen for reproduction into the next generation are selected by their fitness values as well as keeping diversity within the population. After *selection* of the candidates based on fitness and diversity, selected individuals then undergo *recombination* and/or *mutation* to produce new candidate solutions. Recombination produces *offspring* by combining the objective values of *parent* similarly to passing genes from two parents to a child. *Mutation* only involves one *parent* and simply changes specific parameter values as a means of exploring the solution space. The resulting population consist of the best solutions from parents, offspring, and mutations. If an *offspring* or *mutation* does not yield a more optimal solution than the previous generation, it is likely not kept in the population. After reproduction, all of the solutions are evaluated for fitness and then the *selection* process repeats. The *selection* process continues until some stopping criteria is reached such as number of generations completed, performance fails to improve, or acceptable performance achieved.

The basic Evolutionary Process is outlined in Algorithm 1.

In all cases of computation the decision maker (DM) desires that only a few solutions are available for ease of decision. With a Multi-Objective Evolutionary Algorithm (MOEA), the algorithm is attempting to optimize a vector objective function with constraints between multiple conflicting objectives. It is desired that an MOEA generates MOP solutions in P_{true} which provide a trade-off of performance efficiency and effectiveness [14].

MOEAs have a great advantage over other MOP search techniques they are capable of encoding individual solutions in numerous straightforward representations as chromosomal

Algorithm 1 Evolutionary Algorithm Process

Generate the initial population of individuals

Evaluate the fitness of each individual in that population

while Stopping Criteria Not Met **do**

Select the best-fit individuals for reproduction - parents

Breed new individuals through recombination and mutation

Evaluate fitness of new individuals

Select the best fit individuals to be parents for next generation

end while

objective values. Although the No Free Lunch (NFL) Theorem [76] implies that MOEAs are not universally robust solutions for all MOPs, in general the Problem Domain model does not need to be modified for an MOEA. This makes understanding the search process significantly easier as much of the information remains in its native form [11].

Achieving the exact Pareto front for any given MOP is difficult and often not acceptable given reasonable time constraints. MOEAs allow for reasonably good approximations of PF_{true} in limited computational time. The purpose of MOEAs is to find these acceptable approximate Pareto fronts and Pareto optimal solutions within a given error. Types of Evolutionary Algorithm Techniques Include

- Genetic algorithm [14]
- Gene expression programming [58]
- Evolution strategy [11]
- Memetic algorithm [46]
- Differential evolution [165]

- Nueroevolution [164]
- Learning classifier system [171]

2.7 Multiagent Systems

A common design question for any IDS is how to maximize the benefits and minimize the penalties associated with network-based as well as host-based approaches. The MAS paradigm offers a way to accomplish this, with the added advantages of flexibility and robustness provided by this approach.

Russell and Norvig [149] define a single agent through several properties: autonomous operation, ability to perceive the environment, persistence over a long period of time, ability to adapt to change, and ability to create and pursue goals. These goals are typically in support of a broader objective. Franklin and Graesser [62] provide a survey of definitions for software agents, and an associated taxonomy.

Multiagent systems can be used to solve problems that are difficult or impossible for an individual agent or a monolithic system to solve. A multiagent system is a collection of agents that collaborate, explicitly (e.g., via cooperation) or implicitly (e.g., via competition) to achieve a broad objective or series of objectives. The main feature which is achieved when developing multi-agent systems is flexibility, since a multi-agent system can be added to, modified and reconstructed, without the need for detailed rewriting of the application. These systems also tend to be rapidly self-recovering and failure proof, usually due to the heavy redundancy of components and the self-managed features.

In the networking domain, if agents are required to be *mobile*, then all hosts in the network must have a generic agent platform installed which provides the environment in which the agent executes. Agent migration then consists of sending agent state to a remote process responsible for reinstantiating the agent.

Jansen lists some specific advantages of a mobile, agent-based IDS [90]:

- *Overcoming network latency* - if an agent is present on a node requiring remedial action, the agent can respond more quickly than if action must be initiated by a central coordinator

- *Reducing network load* - Communication requirements are reduced by allowing agents to process sensor data locally, instead of requiring each node to send sets of sensor observations to a central processing location. Sharing the results of local processing incurs a relatively light demand on bandwidth.

- *Autonomous execution* - surviving agents continue to operate when part of the IDS fails

- *Platform independence* - agent platforms with standard interfaces may be written for multiple operating systems to allow effective MAS execution in a heterogeneous OS environment

- *Dynamic adaptation* - the system can be reconfigured during run-time in a variety of ways. The mobility of the agents allowing them to seek effective positions is a reconfiguration. Agents can clone themselves or request assistance from other agents in high demand situations. Selected agents can be replaced while non-selected agents continue to operate. One can also update repositories of behaviors and parameters which agents access periodically.

Potential disadvantages include decreased performance and/or increased resource consumption when mobility is implemented ineffectively. Also, since each agent is a member of a trusted network that, if compromised, could provide the attacker considerable leverage, digitally signed communications (including migrations) are essential.

Multiagent systems provide countless research opportunities in the field of intrusion detection, as illustrated by [90], and many other fields of complex problem solving [167].

The three areas of interest for this research effort include distributed systems, autonomous self-organizing agents, and adaptive agents.

When talking about a problem, one might ask, *why distribute the intelligence*? The main reason to distribute intelligence in computer systems is to decompose a complex problem by breaking it into many less complex problems with a subsystem designed to solve each part. If each subsystem performs its action efficiently the problem can be conquered effectively [57]. A home builder does not need to know the process of creating lumber; the builder just needs the wood to build the house. In intrusion detection with multiple agents, an agent does not need to know the exact data another agent calculated, just whether or not the agent located an intrusion and its location [181].

Further rational for the use of distributed intelligence is that the problem itself is physically distributed [57]. A single agent simply cannot adequately monitor hundreds of nodes to detect for intrusions. Multiple agents must *collaborate* in order to monitor the physical distribution of nodes on the network.

Problems are also widely distributed in terms of functions. Not one entity can be the knowledgeable expert for every part of a *complex* system [181]. A race car driver is not an expert in tire production, engine construction, and fuel refinement. It is too burdensome, and even if this driver was an expert in all of these facets, would one expect this well educated driver to individually perform all of these tasks prior to each race? MASs allow *heterogeneous* agents, meaning agents with different specialties or tasks to perform [57], to work together in order to complete a complex task such as network classification.

Collaboration and *self-organization* is required among agents in distributed systems. Besides the heterogeneous nature of problems within a complex network, problems often present complexity in the form of difficult calculations or multiple objectives and constraints [167]. Each agent must take care of its own local problems and communicate the results to its collaborating agents. Focusing on the local problem allows for the parallel

processing of a larger objective, while communicating necessary and important results (summary) to the collaborating agents allows for self-organization in the form of organizing tasks and providing alerts if the agents need to *adapt* to system changes [57].

A great example of collaboration among local entities completing a global task is air traffic control towers. The amount of data, constraints, and parameters with multiple failure points makes a single controller model impossible for maintaining the system of global flights. Instead local air traffic controllers take care of the inbound and outbound flights from their airport, much like an agent monitoring a node, and only communicate necessary information to other towers. One example of vital information that requires communication that leads to *adaptation* and *self-organization*is an emergency landing. If a flight requires an emergency landing, the towers and aircraft must communicate the needs of the system to adapt the flight patterns for an unexpected landing [167].

This scenario is not unlike a network under attack. MFIREv3 is meant to allow agents to adapt to a dramatic change in the environment when a network intrusion takes place. Agents must communicate the effects of the attack and reorganize to classify and address the intrusion.

Another form of *adaptation* within a MAS, besides its role in self-organization, is the possibility of an agent to adapt its own focus [57]. Agents are meant to perform specific tasks efficiently and effectively, however unforeseen changes in the environment could reduce or increase the necessity of an agents primary task [181]. For instance, a set of X agents in the MFIREv3 model each are capable of identifying one specific type of attack. If all of the agents capable of detecting a Worm propagation suddenly failed, the remaining agents would select a subset of agents to take on the feature recognition capabilities of a Worm propagation to prevent the network from the vulnerability of an undetected Worm attack.

For a detailed description of Multiagent Systems and Distributed Multiagent Systems see [57, 90, 167, 181]. The following section details the concepts of reputation and trust among agents in a Multiagent System.

2.8 Reputation and Trust

In the general sense, reputation is *what is generally said or believed about an agent's character or standing* [92]. The pursuit of an adaptable multi agent system achieving Meadows' notion of self organization [124] leads to the need for a way to effectively govern the agents' communication and mobility patterns.

We consider the use of *reputation* to achieve this purpose. Reputation is defined as the collective observation, by a society, of a particular agent's past behavior. A reputation system provides publicly-available assessments of agents' trustworthiness based on ratings from past transactions [155].

Trust, on the other hand, is a subjective internal measure by which a particular agent makes use of the reputation of and its own record of direct experience with other agents to govern its interactions [88].

A variety of trust models exist. Huynh et al. [88] review three distinct modeling approaches:

- Mechanisms deriving trust via certificates, rules, and policies
- Centralized trust mechanisms in which witness observations are collected by a central authority; also known as centralized *reputation* mechanisms
- Decentralized systems

In each model, the agent evaluating the trustworthiness of another is called an *evaluator*, while the evaluated agent is called the *target*. The evaluator may query *witnesses* with direct experience with the target. The witnesses respond to the evaluator's queries with

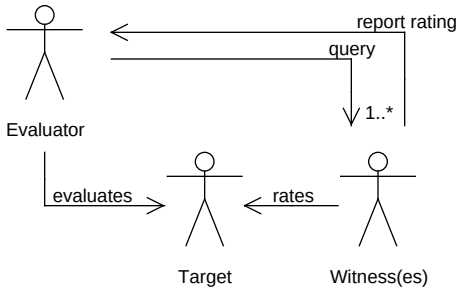

Figure 2.13: Generic trust model: conceptual relationships [77]

ratings. The collective ratings impact the target's reputation, which the evaluator uses along with internal criteria to determine the target's trustworthiness. Figure 2.13 demonstrates a generic view of the relationships between the evaluator, target, and witnesses [77].

Trust and reputation are central to effective interactions in open multi-agent systems (MAS) in which agents, that are owned by a variety of stakeholders, continuously enter and leave the system. Such a concept of reputation focuses on the difficulty for agents to form stable trust relationships necessary for confident interactions. This implies an environment in which individual agents are greedy, able to make their own decisions, and not necessarily seeking to optimize the good of the system.

Many computational and theoretical models and approaches to reputation have been developed [53, 88, 147, 155, 189]. In all cases, electronic persona are created, which reflect the specific forum under evaluation.

MFIREv1 and MFIREv2 relied on a centralized reputation system with the goal of indicating the level of service one agent expects to get from another. The basis of a centralized approach is the same as that of online rating systems employed for shopping [147]. Following an interaction, a witness *conceptually* rates the target according to the perceived level of service received. The rating is stored centrally and combined with

other ratings to allow the centralized evaluator to determine the resulting reputation. This reputation can then be used as a criterion by which other agents or online shoppers decide whether to *trust* the reputation holder.

The first objective of this research effort uses a different definition of reputation. In the commonly described system, agents do not make decisions on their own. Agents simply use the available local observations in order to make a classification, which is sent to a central agent controller. The controller dictates to individual agents whether they should move to a new location. In this sense, agents are simply the eyes of a single central controller and cannot be enticed to perform any individual actions. The elimination of a central controller allows the agents to choose freely their location based on reputation and movement criteria. The movement criteria encourages exploration of the network to areas not under observation by other agents. This feature eliminates the possibility of "piggy-backing" on another agents search.

One example of a multiagent system using Reputation is SPORAS [189], in which new agents start with a minimum reputation value, and build up reputation during their time on the system.

The rules of the MFIREv3 Reputation system are listed in Algorithm 2.

The first two rules discourage users from simply creating new accounts to escape the consequences of a series of bad interactions. But one can imagine environments in which migration should be encouraged, such as when the service an agent can provide is dependent on the agent's location. In such cases, a migration threshold may be set below the restart value. In this way, reputation may be used to govern the mobility patterns in the multi agent system. This is one of the desired behaviors of our system and is thus part of the movement model.

Other research contained in [88, 147, 155] demonstrate other approaches.

Algorithm 2 MAS Classification

1) New users start with a minimum reputation value, building up reputation during their activity on the system

2) The reputation value of a user never falls below the reputation of a new user

3) After each transaction, the reputation values of the involved users are updated according to the feedback provided by other parties, which reflect their trustworthiness in the latest transaction

4)Users with very high reputation experience much smaller rating changes with each update

5) Ratings must be discounted according to age so that the most recent ratings have more weight in the evaluation of a user's reputation

We can consider the concept of Reputational Incentives defined in [34]: the truster calculates the reputational gain (or damage) that a trustee experiences as a result of good (or bad) feedback being communicated to the society, and considers this as an additional incentive. Pertaining to the intrusion detection problem, a trust model is defined by [19], and also makes use of the NetFlow concept of flows.

2.9 Defensive Measures of Network Agents

Automated defense of a network intrusion can provide another barrier for intrusions set on disrupting a network. A wide range of automated defense techniques exist of varying effectiveness, complexity, and also ethicalness [127].

The Department of Defense (DoD) conducted a 90-day exercise consisting of 30 cyber defense companies on the issues pertaining to automated defense [127]. The conference focused on the issues pertaining to the prevention of attacks as opposed to the current response techniques common in network security. A major issue with preventing attacks is

the legality and ethics with striking a threat before or after an attack with a cyber attack of one's own [158].

The present state of network security consists of what we shall refer to as *passive defense*. The defensive measures do not engage an attacker outside of the network itself. Instead passive defense aims to respond to a known intrusion and stop it, while possibly gathering information on the attacker location [9].

One common form of passive defense is the *Honey Pot*. A Honey Pot is a trap set to detect, deflect, or in some manner counteract attempts at unauthorized use of information systems. Generally it consists of a computer, data, or a network site that appears to be part of a network, but is actually isolated and monitored [157]. A Honey Pot seems to contain information or a resource of value to attackers, but the information is merely a distraction to monitor the attacker for possible location [127].

Another form of automated attack defense is *rate limiting*. Rate limiting helps preventing flooding of from Denial of Service (DoS) attacks. According to many tests, rate limiting is a formidable reactionary response against flooding DoS attacks [132]. Rate limiting is simply the application of rules to a network to limit traffic flow. Since the purpose of a DoS attack is to flood the network with malicious traffic, rate limiting prevents the attack from spreading outside the compromised node quickly. This makes targeting the correct node quickly a much easier task, and it also prevents the destruction of network resources. While the rate limiting rules are applied however, the rate at which normal traffic passes through the network is hindered as well [83].

Many pieces of literature propose other reactionary defense mechanisms such as killing of active network connections, filtering, reconfiguration, re-imaging, artificial immune systems, and source-traceback mechanisms [83].

Killing of active network connections of infected nodes is a quick method of containing the attack. In most cases this simple measure can immediately stop a Denial of

Service attack or a Scan [9]. In cases of Worm propagation attacks, the "flood gates" must close all around every infected node on the network or else the Worm simply propagates from the areas still connected to the network [9].

Reconfiguring the network often confuses many automated attacks as well as human attackers. Certain reconfiguration measures can help force an attack into non-critical areas of the network, away from sensitive data, and into Honey Pots [6]. Re-imaging infected nodes simply sets the node back to factory defaults and removes the infection [127].

Artificial immune systems are a growing field in network defense. Immune systems work to by learning about attacks as they occur in order to build up "immunity" in future cases [158]. The procedure also works to repair attacked and infected system nodes by re-imaging nodes to get them on-line and stop the spread of malicious traffic [158].

None of these passive reactions to network intrusions, including rate limiting and source traceback mechanisms, are considered ethically unsound. All of these defenses are metaphorical pieces of armor, with the exception of source traceback mechanisms, designed to withstand the bullets of malicious attacks. The source traceback mechanism is simply a locater to find out where the "bullets" originated. These mechanisms provide no threat to the outside Internet as all of the defensive procedures are internal to the network.

The idea of defensive measures being internal to one's own network follows the ideology of safeguarding owns home or other property. In general, safeguarding one's property is encouraged, however leaving the property after it the attack in order to seek retribution from the attacker might cross certain legal and ethical boundaries [158].

Active defense is a relatively new concept in network security, although it stems from an idea as old as time [83]. Network administrators of high-value companies wish to do more than build up the metaphorical armor of network security. Armor wheres down under a constant barrage of attacks, and even the best armor has vulnerabilities. It only takes

locating one vulnerability for an attacker to render the rest of the defensive measures useless [9].

This is the plight of network security administrators. Attackers constantly barraging their systems with attacks in hopes of finding a vulnerability, but the network being attacked is not allowed to strike back. Certainly attackers would be less willing to go after a system if it was known that the network security team, or automated agents, would be allowed to fight back with malicious code as well.

The broad definition of *active defense* is any measure originated by the defender against the attacker. The purpose of any computer network defense is to protect information systems [158]. These active measures should at least thwart any attack in progress, and ideally make further attacks more difficult. We can divide them into three broad categories: counterattack, preemptive attack, and active deception [83].

Active deception directs the attacker to a virtual model of the network or a Honey Pot [83]. The attacker believes the attack is destroying the targeted system when in fact the attacker is no longer a direct threat. Active deception is effectively an active Honey Pot, where the defender actually attempts to push the attacker towards the trap as opposed to letting the attacker stumble into it. This type of defense does not provide any ethical dilemma, since all of the action still takes place in the defenders own network [157].

The ethical dilemma is reached with counter-attacks and preemptive attacks to a network. Much like their names suggest, a counter attack is a direct attack by the defender on the attacker's network during or immediately following an attack [83]. A preemptive attack is one where the defender knows the attacker is preparing for an attack and strikes the attacker's system before the attack can take place.

The idea of preemptive attacks presents the greatest amount of skepticism as a morally acceptable defensive strategy. Attackers are not likely to advertise that they are going to attack a specific network, with certain exceptions in cases involving Anonymous, LulzSec,

and similar groups [127]. Having malicious code on ones one computer is not illegal, and therefore the only crime committed in a case of preemptive attack is by the "defender" [127]. Even if a group such as Anonymous states that they are going to attack a specific company, the IP addresses of the attackers is unknown unless the company was already illegally searching for the group. Preemptive attacks in the name of defense follow the same logic as attacking a man at his home because he threatens to hurt you the next day.

The main difference however, between attacking a man at his home and attacking his network resources is that in the former case, laws, police, and the justice system are capable of handling physical disputes. In many recent cases of network attacks, there does not appear to be any sort of law enforcement capable of deterring criminals. Confusing international laws and no real effective "Internet police", leave companies with viable assets wondering how to protect themselves [127, 157].

This "cyber wild-west" makes the morality of counteracts against known attackers a debatable discussion; although the illegality remains concrete . Counter attacks involve locating the attacker's network resources and attacking them with a Worm or DDoS the same way a normal attack would take place. In some cases, the counter attack may try to install programs that help discover the physical location of the attacker in order to help prosecution for criminal offenses [127]. Regardless of ethical and legal issues pertaining to counter attacks, commercial industries with valuable or sensitive data are resorting to these measures and if the "Internet police" are unable to handle hackers in their mother's basement, it is unlikely that they are able to catch the professional security officers as well [127].

2.10 Summary

This chapter considers, in Section 2.1, the Autonomous System level Internet topology and traffic modeling requirements in order to conduct the desired objectives. This is followed in Section 2.2 by a discussion of the prototypical Pattern Recognition system as a

template for what our system must implement and accomplish. In particular, our research implements a classification system. Evaluation of classification systems is consequently discussed in Section 2.2.7. Section 2.3 examines pattern recognition in the context of identifying malicious network activity, which is known as intrusion detection; particular consideration is given to flow-based techniques. In Section 2.7, a brief discussion of multi agent systems and their applicability to intrusion detection is presented. To achieve a degree of 'self-organization' as defined in this list, Section 2.8 considers the notion of reputation. A far broader way to achieve self-organization via the use of Evolutionary Computation is presented in Section 2.6.2. Also, the evolving definition of Multi-Objective Optimization emergence is addressed in Section 2.6. Outside the realm of detection, identification, and self-organization, section 2.9 presents measures taken by networks and agents to aide in the protection of the network.

The next two chapters employ these concepts in the presentation of multi agent system designs for detecting, classifying, and countering network attacks.

III. MFIREv3 Design Methodology and Implementation

An updated MFIRE design is required per the goals and objectives of section 1.2 for autonomous classification of network attacks in a model type network. This research builds upon the MFIRE framework of Hancock [77] and Wilson [185] by adding additional attack features, introduced in section 3.2.5. MFIREv3 implements an improved attack classification system with feature selection in section 3.3. Furthermore, in Section 3.5 we integrate features proposed by Holloway [84] to allow agents to behave in more elaborate ways and take a roll in the defense of the network. This chapter develops these design concepts.

Section 3.1 formalizes the problem of Intrusion Detection we solve with this research effort with Section 3.2 detailing the design of the simulation environment. The methods for agent classification training are deigned in Section 3.3. The design methodology of testing the defensive aspects of agents in the network is discussed in Section 3.5. A short discussion of the visualization environment and a summary conclude the chapter.

3.1 Intrusion Detection System Formalization

Intrusion Detection (ID) system design is an ongoing process. Simulating network environments and traffic creates controlled scenarios providing insight into detection capabilities of agents. The multi agent system, with several performance-enhancing details, is leveraged in this design in order to maximize the performance. The agents are designed to be mobile and cooperative in terms of sharing feature observations and defense. Over a series of simulated attacks, MFIREv3 searches for optimal agent locations for both effective detection and defense of attacks.

The design of a suitable network simulation environment involves the representation of essential network components and operations. Specifically, nodes must route traffic,

generated by processes, over links with limited capacity, in a topology reflective of what is seen in the real Internet (see Section 2.1). Some of the processes represented are *normal*, generating traffic according to distributions seen on the real Internet, while other processes represented are *malicious*, causing congestion on network links, systematically extracting information regarding potential vulnerabilities of network nodes, or spreading copies of themselves to other nodes without authorization.

Enabling the properties described in such a simulated network environment requires a both representation of traffic as content-bearing packets as well as facilities for delivering these packets to specific destination processes. Many facilities for instantiating a network complete with its nodes, links, processes, and properties of each already exist and are utilized in this research. To aide in the understanding of high-level traffic patterns, Intrusion Detection, defensive measures, and agent movement, visualization facilities must be considered as well.

3.2 Simulation Environment

The package hierarchy provides a framework in which to place the required representations of MFIREv3. In addition to the network simulation environment, a multi agent classification system is designed as a set of processes, with components including agents and an optional agent controller. To support the agents' classification responsibilities, interfaces are designed for classification techniques and feature definitions, enabling changes in detailed implementations without requiring changes to the system architecture. Lastly, the defense system is supported by the MAS and classification system to provide supplemental support for the elimination of network threats.

A detailed History of the MFIRE design, from its origin in SOMAS [84], is located in Appendix A.

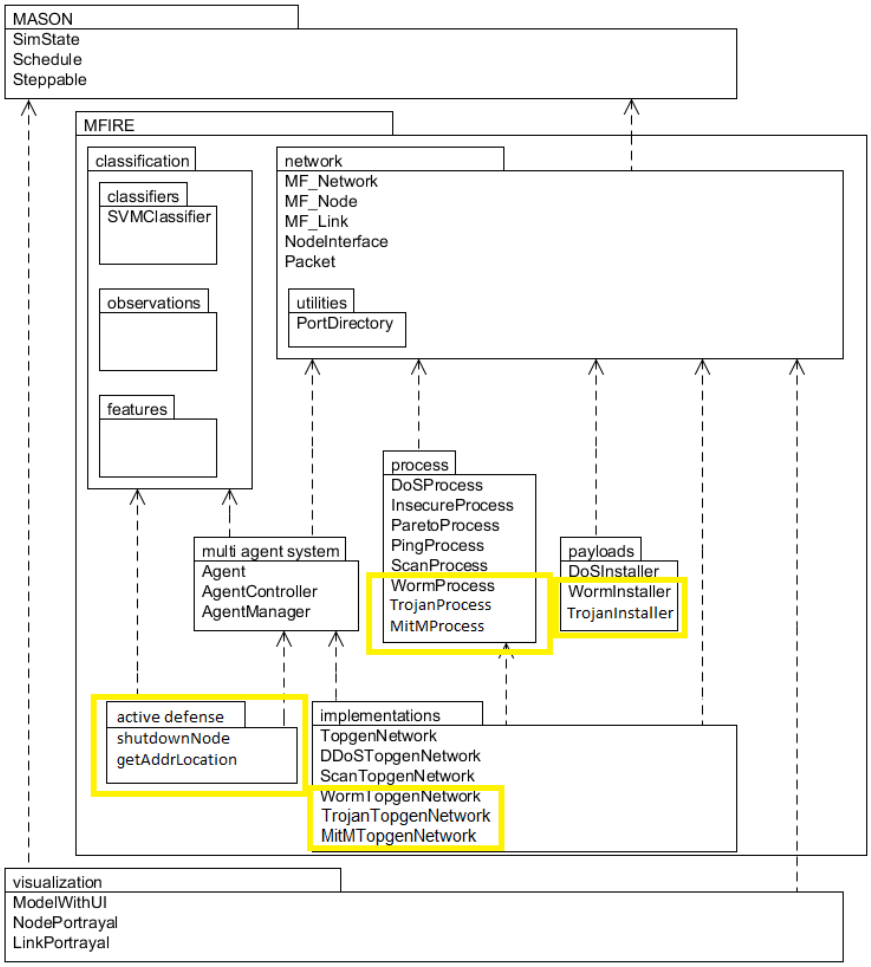

Figure 3.1: MFIRE v3 Package Diagram.

Figure 3.1 presents a general view of the package hierarchy involved in the simulation. Highlighted regions are additional implementations to MFIREv3. Also, the agent controller and manager are not used in this version, however they remain in the framework. The controlled, one-way dependencies between the visualization layer, the *domain layer* (MFIRE), and the *application layer* (MASON) exhibit a software engineering principle known as *Model-View Separation* [106]. This principle states that domain objects should not have direct knowledge of view (UI) objects. It allows the visualization layer to be

changed without requiring any changes in the domain or application layer. The domain layer (MFIRE) consists of the following groups of classes:

- Network - includes representations of physical domain entities of interest. This is the 'core' of the simulation.
- Scenarios - concrete realizations of the abstract MFNetwork. The prominent class is the TopgenNetwork, which includes facilities for loading a network produced by the Topgen AS-level Internet topology generator. Each class in this package is characterized by a a unique set of Processes initially running on a subset of the nodes.
- Processes - These are analogous to the networked applications on the real Internet. Each Process runs on a host node and may receive and/or generate traffic.
- Payloads - Specially crafted payloads execute code when opened by a certain receiving processes. These payloads can be written for legitimate purposes, such as Remote Procedure Calls (RPC), but our focus is on payloads that install malicious processes on the receiving node.
- Multi agent system - This package includes the "worker bees" - the Agents, the "queen bee" - the AgentController, as well as AgentManagers with special local oversight of any Agents on the same host node. In this version of the MFIRE system, the Agent Controller and Managers are bypassed by the agents direct communication in a fully distributed system, however the Controller and Managers can be reinstantiated with minor changes to the test.
- Agent Defense- This package allows agents to take an active roll in eliminating network threats. The current version includes the ability to shut down a compromised node, limit traffic flow, and gather information of the attack location if available.

- Classification - Agents make use of entities in this package to make local classification decisions. Included are the classification algorithms, enclosed in the 'classifiers' package, and the observations and features used. Strictly speaking, both observations and features are statistics-based calculations, but we distinguish the observations as being more "raw" than the features. By 'feature' we imply there is something composite in its nature - it may be an average of observation values or the result of some other series of mathematical operations on the observations and/or other features.

At the top of Figure 3.1 is the MASON discrete event simulation engine package. MASON provides many facilities for the execution of the simulation as well as the visualization tools. The details of the visualization are specified via entities in the visualization package at the bottom of the diagram.

Figure 3.2 shows the class diagram of MFIREv3. Architectural detail of some of the prominent classes is provided, however it is not a comprehensive listing. The diagram provides some of the essential class associations between the agents, provider agents, attacks with the MFIREv3 network simulation.

3.2.1 Network Design.

From Section 3.2.1 we discuss how network topology is the arrangement of various nodes within a computer network. There is both a physical and logical topological structure, where the physical model illustrates the locations of the nodes and their interconnections while the logical model illustrates the data-flow between components within the network. The physical network components simulated in this research investigation include [77]:

- Nodes - each node represents an Autonomous System (AS). Internal to an AS is a collection of routers, switches, firewalls, and edge devices, including servers and

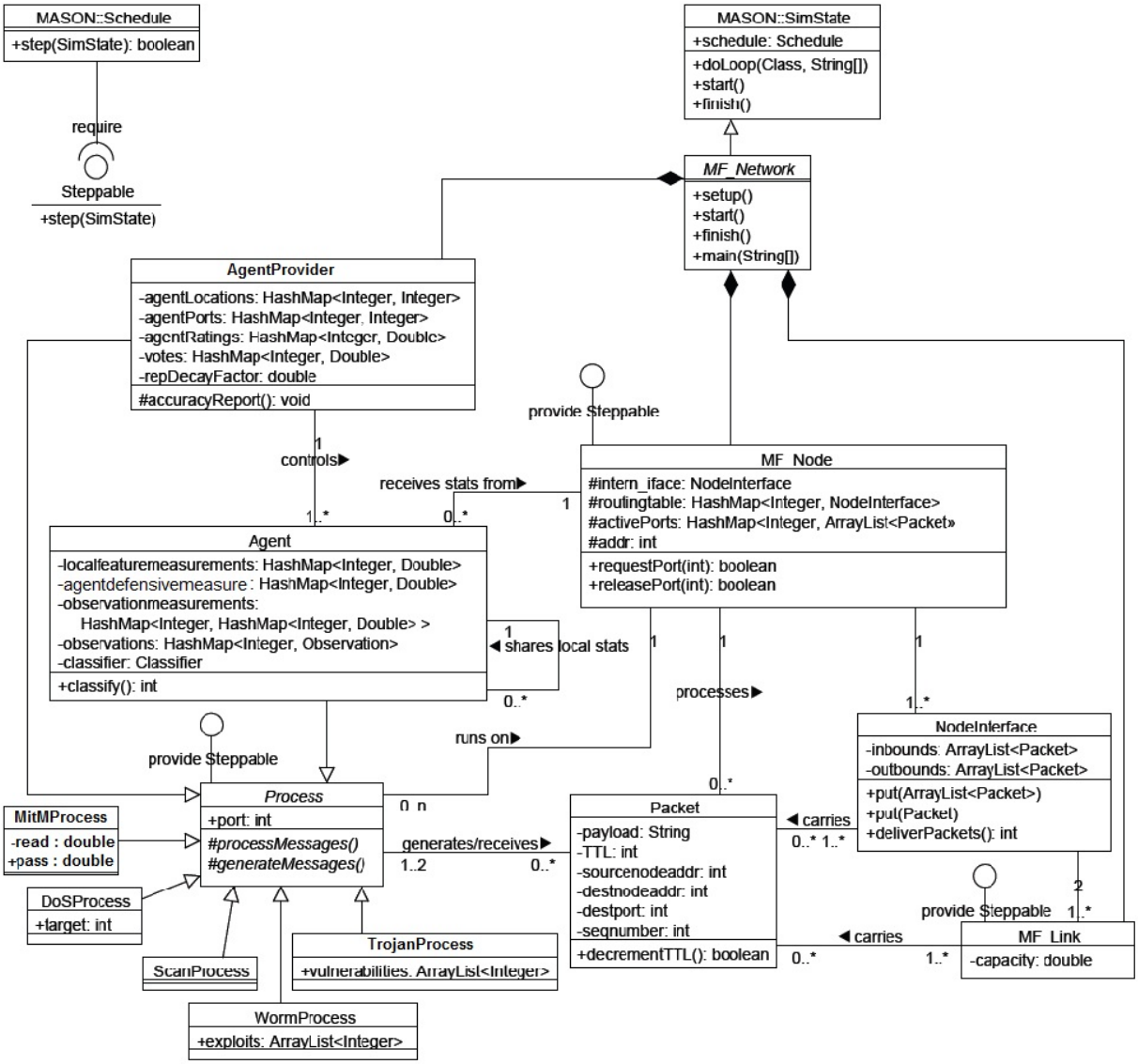

Figure 3.2: MFIREv3 class diagram.

clients. These devices are all abstracted into one node in our simulation, represented by the MF_Node class in Figure 3.2. Nodes route traffic via routing tables, initialized via the Floyd-Warshall shortest path algorithm [61]. This is analogous to gateway routers employing BGP on the real Internet, though with BGP, policy decisions often trump routing efficiency (competing Internet service providers, for example, may

refuse to allow 'through' traffic without compensation). Each node is addressable by a unique identification number. Nodes provide resident processes with basic communications facilities, such as the `send()` method, which creates and sends packets. Nodes implement the `Steppable` interface and therefore supply a `step()` method invoked on each timestep of the simulation. This method primarily switches packets from the inbound queues of all NodeInterfaces to the outbound queues of NodeInterfaces identified in the routing table, via lookup on the packet's destination address.

- NodeInterfaces - These are intermediaries between Nodes and 1) Links; or 2) Node's resident Processes. The first case includes all external-facing interfaces, while the second describes the Node's internal interface. Each is an entry/exit point. All NodeInterfaces have an inbound queue and an outbound queue. The inbound queue is read by the attached Node and written to by the attached link. The outbound queue is read by the attached link and written to by the attached Node.

- Ports - associated with nodes, ports are the communication end points for processes running on servers and clients. In the real world, each computer typically has many thousands of ports associated with each transport-layer protocol. For example, there are 2^{16} ports available for Transmission Control Protocol (TCP) and another 2^{16} for User Datagram Protocol (UDP), the number being fixed by the width of the port field in the segment, respectively datagram header [143, 144]. In our simulation, each port on an AS node corresponds with a port on an arbitrary host internal to the AS.

- Port Directory - Certain "well-known" ports are reserved for special purposes. This is the case with the real Internet, for which a list is maintained by the Internet Assigned Numbers Authority (IANA) [2] specifies how certain ports are to be used, such as port 80 for Hyper Text Transfer Protocol (HTTP) traffic. When these standards are

adhered to, finding public services is greatly simplified. Also, filtering of certain expected types of traffic becomes simple. Observe that, in our simulation, some ports are reserved for components of the multi agent system.

- Links - links in our network simulation are strictly point-to-point and connect autonomous systems together. Links are full duplex but have finite bandwidth. Depending on the scale of the simulation, links may vary in length, affecting propagation delay. One of three scales is specified at the start of each simulation:
  - LOCAL - All links have the same unit length. Packets traverse these links in one step of simulation time.
  - REGIONAL - Link lengths vary from one to ten units. This is useful when the simulated AS topology spans a continent.
  - GLOBAL - Link lengths vary from one to 100 units. This is appropriate for simulation of an AS topology in which some of the nodes are satellites in geostationary orbits, for which propogation delays can indeed be on the order of 100 times those of terrestrial links.

  Scale is realized with each link being composed of sublinks. Links implement the `Steppable` interface. Each timestep, when the Link's `step()` method is called by the `Schedule`, the Link causes each Sublink to pass its traffic to its adjacent Sublink (or, ultimately, NodeInterface).

- Processes - these include processes that strictly generate traffic for the benefit of the simulation as well as classifying agents that generate actual communication traffic (primarily to share observations). All processes run on nodes and must be assigned a port before they can send and receive packets. Processes implement the `Steppable` interface. When `step()` is called, the Process first receives and processes traffic, and then generates outbound traffic.

- Packets - Each packet consists of the following:
  - Source node address - identifies the Node of origin
  - Source port - the port used by the sending Process
  - Destination node address - identifies the Node hosting the intended recipient Process
  - Destination port - communication endpoint for the intended recipient `Process`
  - Sequence number - Facilitates sending messages spanning multiple packets
  - TTL - Time To Live - the number of hops allowed before some intermediate Node discards the packet. This mitigates problems arising from routing loops induced by congestion or misconfiguration of the routing tables.
  - Payload - a string containing the message the sending Process wishes to pass to the intended recipient. The format of this message is entirely up to the communicating processes.
  - size - Indicates the size of the payload, in numbers of characters, if a real payload is used. If a real payload is not required (e.g. to simulate background traffic or junk traffic sent by denial-of-service processes), the sending Process can simply specify the desired size of the packet to be sent, leaving the payload string null and preserving memory.

During initialization, after all network components have been instantiated, all Processes, Nodes, and Links are scheduled to execute associated tasks on each *timestep*. They are prioritized as follows:

- 1. Processes handle received traffic and generate new traffic
- 2. Nodes handle traffic by switching packets from inbound queues to appropriate outbound queues or ports

- 3. Links move traffic along component Sublinks toward the NodeInterfaces on either end

3.2.2 Multi Agent System Design.

Section 2.7 proposed that the common design question for any Intrusion Detection System (IDS) is how to maximize the benefits and minimize the penalties associated with network-based as well as host-based approaches. This section discusses our methodology in maximizing the benefits of a Multi-Agent IDS while minimizing the penalties using a distributed system.

Figure 3.3 presents a high-level view of the nominal flow of execution from the perspective of the MAS. Five states are shown. Figure 3.3 indicates that the transition from each state is governed by the clock. This implies synchronization among participating elements. Typical message exchange for each state is shown in Figure 3.4. Figure 3.4 is similar to the message passing scenario introduced by Hancock [77], however the central controller is replaced by multiple agent providers shown as Agent P.

The explanation of MFIRE's high-level states is made simpler by assuming agents have been collecting observations from their respective host nodes for nearly a full cycle when it comes time to check in with their Provider. Furthermore, each agent is assumed to have a reputation stored with their Provider.

- Check-in: Agents notify the provider of their intention to participate in the next round of observation exchange and classification. The provider notes the source address and port of each CHECKIN message. Each agent must have a provider, however not each agent is required to be a provider.

- Transition: The provider makes an observation sharing assignment for each Agent that checked in. It does this by constructing a roulette wheel from the reputations of

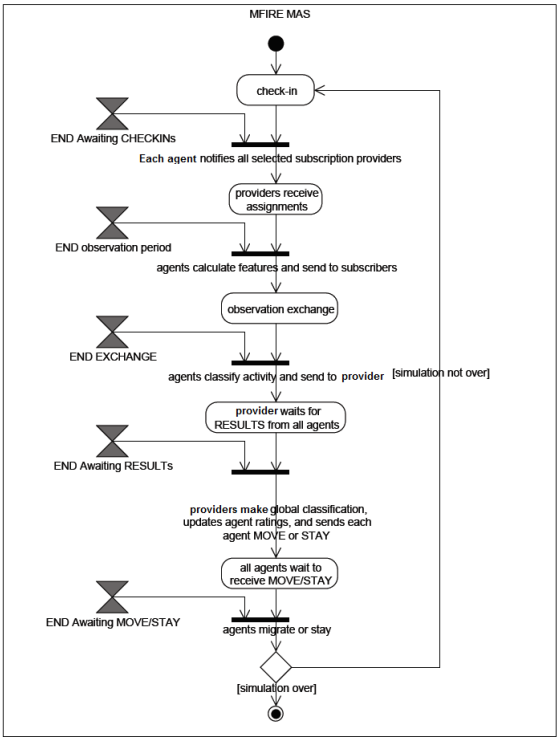

Figure 3.3: MFIREv3 Activity diagrams for the Agent

other checked-in Agents. This roulette wheel is used to make a sharing assignment stochastically with preference given to Agents with higher reputations. The provider notifies the selected Agent with an ASSIGN message.

- Assignments: Selected Agents receive assignments. Some Agents may receive multiple sharing assignments, while others receive none. For each assignment received, the Agent stores the address and port for the designated recipient as contained in the ASSIGN message.

- Transition: End the current observation cycle, calculate features, and start a new observation cycle. Observations are traffic statistics collected on each timestep. At the end of each observation cycle, there exists an Observation set for each traffic statistic measured. Features typically summarize one or more of these Observation

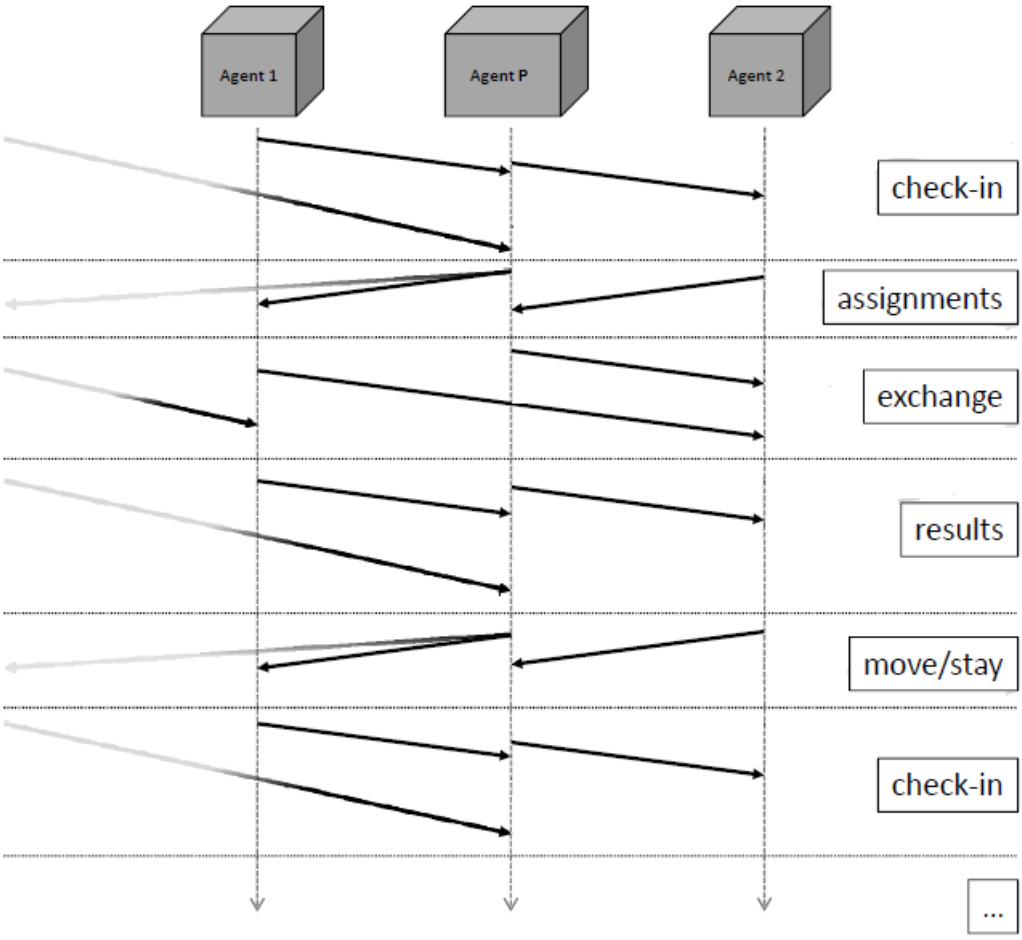

Figure 3.4: MFIRE Activity Diagram Message Exchange

sets. Agents calculate Feature values and store them for later use. Any Agents with sharing assignments also send their set of Feature values to all assigned recipients using SHARE messages.

- Observation Exchange: Agents wait to receive SHARE messages. Each Agent expects to receive one.

- Transition: Agents use two classifiers to make two classifications for the network activity observed over the previous cycle. One of these uses only locally calculated

feature values, while the other uses the combined set of local and received feature values. Agents send the results to the provider agent in a RESULTS message.

- Results: The provider agent receives RESULTS messages from all checked-in Agents.

Algorithm 3 MAS Classification

denote classification by agent a_i at time t using only local feature values as l_{it}

denote classification by agent a_i at time t using combined local and shared feature values (e.g. from peer agent a_j) as c_{it}

denote the majority classification at time t as m_t

denote network activity classes as $\mathcal{A}_k \in \mathcal{A}$ for $1 \leq k \leq K$

denote the vote tally for network activity class $\mathcal{A}_k$ at time t as v_{kt}

Require: $0 \leq \theta_l \leq 1$

procedure MASClassification(θ_l)

for all received RESULTS messages $results_{it}$ **do**

if $results_{it}$ contains a combined classification c_{it} **then**

add 1 to the vote tally v_{kt} for $\mathcal{A}_k$ for $k = c_{it}$

else

add θ_l to the vote tally v_{kt} for $\mathcal{A}_k$ for $k = l_{it}$

end if

end for

$m_t = k : v_{kt} = \max_h v_{ht}$ where $1 \leq h \leq K$

return m_t

end procedure

- Transition:
  - The Providers tally the votes. In each RESULTS message, the vote is the classification made using the combined local and shared feature value sets. When this is not available because the Agent never received a SHARE message, the Provider uses the classification made using only the local feature value set, weighted for less influence. The system's classification is the majority vote.

See Algorithm 3, in which θ_l represents the weight of a classification derived from local feature values only.

  - The Providers update each Agent's reputation as introduced in section 2.8. For each Agent, each sharing assignment it had garners a rating which can positively or negatively affect the reputation. Every Agent furthermore has its reputation decayed regardless of whether it had a sharing assignment, and regardless of whether it checked in. See Algorithm 11.
  - The Provider sends each Agent in its subset a STAY or a MOVE instruction based on whether the Agent's reputation is above or below a threshold.

- Wrap-up: Agents wait to receive MOVE or STAY. Upon receiving MOVE, an Agent selects a neighboring node based on Max Cover, last visited, and proximity to area of interest (possibly attacked or vulnerable node). It also sends a MIGRATE message to the node's it may provide for and updates its location for its provider as well.

Figure 3.5 shows the flow of execution of the Agent and the Provider independently. From this figure it can be deduced that when an agent is acting as a provider it has merely two states: it is either waiting for Agents to check in, or it is waiting for the Agents to send their results, with significant actions taking place on the transitions between states as described above. Meanwhile, a search Agent has a collection of synchronization-related states, and three of the nominal states described above. It is either waiting for an ASSIGN message from the providing agent, or it is waiting for a peer to send a SHARE message, or it is waiting for a MOVE or STAY message from its provider.

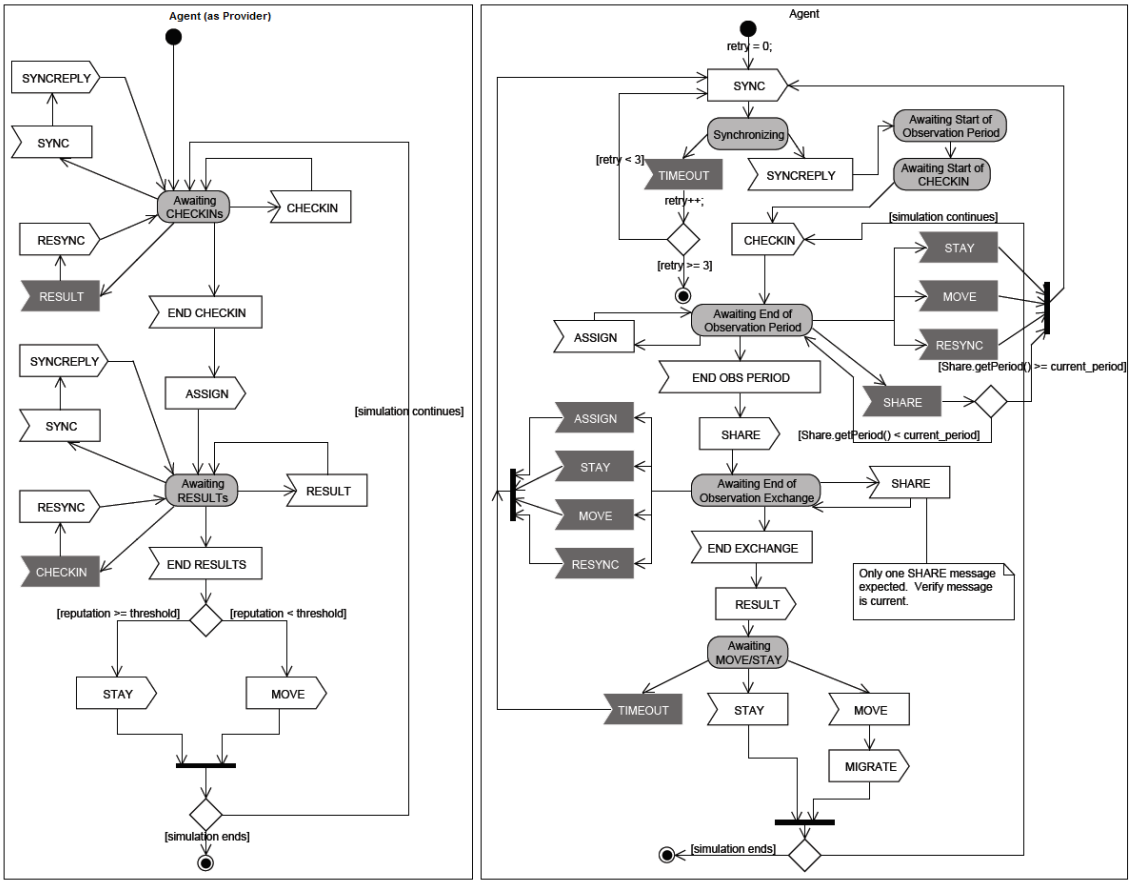

Figure 3.5: MFIREv3 detailed activity diagrams for the agent provider and the agent

Table 3.1: Comparison of MFIRE Iterations

	MFIRE	**MFIREv2**	**MFIREv3**
AS Network Scale	'local' only	'local', 'regional', 'global'	'local', 'regional', 'global'
AS Network Size	10 nodes	100 nodes	100 nodes
AS Network Topology	manually designed	produced by Internet topology modeler	produced by Internet topology modeler
Node Behavior	restricted processing capacity / shut down under heavy load	unrestricted processing	unrestricted processing
Packet Payloads	simulated quantity only	payloads used for interprocess communication	payloads used for interprocess communication
Attacks	DoS	DDoS, Worm, Scan	DDoS, Worm, Scan. MitM, Trojan
MAS communications	out-of-band, instantaneous	in-band with network-based delays	Fully Distributed, in-band with network-based delays
Feature Selection	wrapper method/ Bhattacharya	none	MOEA
MAS Objective	Identify Source and Target of DoS Attack	Identify Type of Attack	Identify Attack and Provide Active Defense
SVM Kernel	Gaussian	Gaussian	Linear and Gaussian

Table 3.1 summarizes the key differences between this research, the system that is quantitatively tested by David Hancock [77], and the system implemented by Timothy Wilson [185]. In general, MFIREv3 features similar networking as in previous experimentation, with improved feature selection, improved classification, full distribution of agent communication, increased attacks, multiple attack classification, and defense features.

3.2.3 Simulated Network Traffic.

For this effort we utilize our own representative data, known as *Synthetic Data*, as opposed to real world data. Synthetic Data is production data that is not obtained directly from an existing network [59].

Multitudes of reasons exist for choosing Synthetic Data over real network data. Primarily, synthetic data is generated to meet specific needs or certain conditions that may not be found in the original, real data. The DARPA Dataset for Intrusion Detection is commonly criticized by experts as being an "outdated dataset, unable to accommodate the latests trends in attacks" [170]. It is based on real-world network intrusions, however the attacks no longer reflect the same statistics desired in training a Multi-Agent System.

Newer datasets do not contain adequate attack scenarios either. Instead, the data reflects more normal conditions as opposed to the attacks necessary for training agents [73]. University of California Irvine's data repository presents many datasets that are out of date or do not reflect the types of attacks necessary for our research.

Finally, many datasets are proprietary, classified, or incomplete. The anti-virus software companies and government agencies keep very important records of intrusion data, however releasing the data for public interpretation could lead to a rapid advancement in attacks [10]. Quite simply, the most accurate information on current network attacks is not available for public use.

Since none of the current repositories present the adequate datasets necessary for our research we must instead create synthetic data as is done in many research efforts concerning fraud detection, data mining, confidentiality systems and as is our case, intrusion detection [170]. The algorithms and overall effectiveness of the system therefore represents its ability to interpret the synthetic data. One cannot ascertain that the same accuracies exist when placed on a real-world network, however the synthetic data is meant to model existing traffic.

The purpose of using synthetic data is to test the algorithm's effectiveness in a controlled environment, but the environment itself needs to be as close of a model to real attacks and network flow as possible for the test to have any meaning. Fabricating network traffic involves creating background traffic for a normal "base" system, and manipulating the base traffic for each attack according to what effect the attack would have on network traffic [107].

As stated above, the normal (base) mode consists of only background traffic. For this the Pareto model [100] described in Section 2.1 is used with parameters $\alpha = 2.0$, and C ranges from 0.01 to 0.1, randomly selected prior to each simulation. All other attacks models also use this background traffic with additional traffic reflected in their attacks.

The fabricated synthetic data, following the Pareto model, adequately reflects real world network traffic to the necessary degree for testing our simulation and showing proof of effectiveness [100].

3.2.4 Observations and Features.

Each observation in MFIREv3 represents a traffic statistic collected over the duration of a single time period. These are used to derive feature values, which are the average and standard deviation of the observations within one observation period. We take inspiration for flow metrics from both Cisco NetFlow [187] and Moore [133], with emphasis on implementing metrics applicable to *microflows* (see Section 2.4). The fourteen metrics

defined here represent a cross-section of possible flow-based statistics, but future work should examine additional metrics, including implementing a macroflow approach (see Section 2.4 and [133]).

The fourteen observations collected by agents in MFIREv3:

1. Average number of bytes per $< destaddr, destport >$-tuple
2. Average number of bytes per $< sourceaddr, sourceport >$-tuple
3. Number of distinct destination addresses
4. Number of distinct $< destaddr, destport >$-tuples
5. Number of distinct destination ports
6. Ratio of destination ports to destination addresses
7. Total number of inbound bytes
8. Total number of inbound packets
9. Ratio of packets to $< destaddr, destport >$-tuples
10. Ratio of packets to $< sourceaddr, sourceport >$-tuples
11. Number of distinct source addresses
12. Number of distinct $< sourceaddr, sourceport >$-tuples
13. Number of distinct source ports
14. Ratio of source ports to source addresses

Clearly there are many linear dependencies in this set of observations. Care must be exercised when performing feature selection from this set. When decomposed into the

84 combined features, the Bhattacharya Coefficient cannot effectively eliminate the linear dependencies. This is illustrated in Figure 3.6, which separates the features by thousands of bins and still fails to eliminate the dependencies. It is for this reason a Support Vector Machine is required for classification in order to take features to a higher order. Details on general feature selection are provided in 2.2.2, and the feature selection process utilized in the effort is examined in 3.3.3.

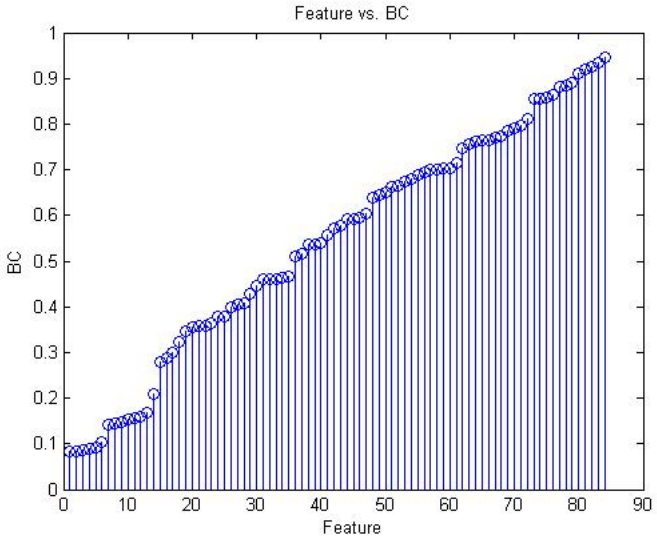

Figure 3.6: Illustration of Linear Dependecies in Features

Additionally, linear separability of all 5 attacks and the normal state given a set of three features selected by the MOEA for performance proved futile. Figure 3.7 illustrates the overlap that would certainly lead to high misclassification rates for the agents given the same features. Two solutions for this are examined in this research: using a higher order, non-linear (Radial Basis Function (RBF)) classifier which was utilized in MFIREv2, and testing multiple feature sets to linearly discriminate between two or more states of the network. The latter approach would provide the ability to give agents different feature sets in an attempt to improve the speed of classification by using a linear classifier.

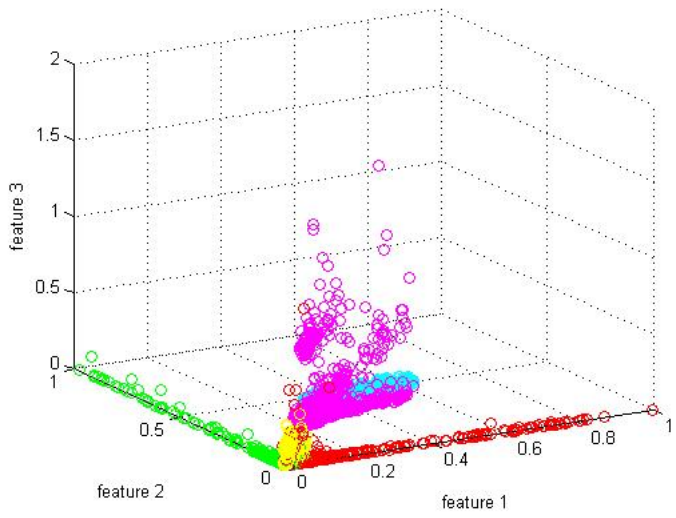

Figure 3.7: Illustration of Non-Linear Separability: Magenta-DDoS Worm- Red Scan-Green MitM-Yellow Trojan-Blue Normal-Black(Covered); Feature 1: Number of distinct destination addresses Feature 2: Total number of inbound bytes Feature 3: Std. Dev-Ratio of packets to (dest addr, dest port)

Figure 3.7 highlights the overlap of attack features using the same set of three features. Each color represents one of the five attacks. There is no linear separability between attacks using these features, however it is possible that using specific feature sets for each attacks could improve separation.

3.2.5 Attack Models.

This research consists of modeling five attacks: DDoS, worm propagation, vulnerability scan, Man-in-the-Middle attack, and Trojan, and one normal (non-attack) mode. These attacks are described in Section 2.5 and represent a broad section of network intrusions. In all cases, background traffic is flowing on the network, and is the predominant source of packets. The attacks implemented in the current research are designed primarily to test to the effectiveness of the MAS reputation system. They are in no way a comprehensive suite of possible malware, but they represent a number of common attacks that both spread

across networks or remain static within the network. Additional attack models should be explored in future research.

The normal (non-attack) mode consists of only background traffic. For this the Pareto model described in Section 2.1 is used with parameters $\alpha = 2.0$, and C ranges from 0.01 to 0.1, randomly selected prior to each simulation. All other attacks models also use this background traffic.

The DDoS attack consists of N processes which flood a single target T with packets to port p at rate r packets per timestep. N, T, p and r are selected randomly prior to each simulation. The node locations of the DDoS processes are random, selected from any nodes in the network. Algorithm 4 illustrates the DoS process which uses a flood of small packets. Using the smaller packets increases the likelihood that the packet is forwarded. Figure 3.8 shows the effect of the attack within the MASON simulation environment.

Algorithm 4 DoS Attack Algorithm

```
Select target
Instantiate Source nodes with DoSProcess
for (All Source Nodes) do
    Determine packet size 1/1000 of link capacity
    while Time not complete do
        Send 1000 packets to Port P of target.
    end while
end for
```

Worm attacks are implemented by a set of vulnerable processes running on a subset of nodes in the network. A worm process is equipped with a single exploit that targets a single vulnerability. If the exploit matches the vulnerability on the target node, the worm is able to instantiate a copy of itself on the target. Worms do not scan for vulnerabilities before attempting the attacks; they simply make an attempt. However, the worm process never sends an attack to a non-existent node. This is only possible if the worm has previously

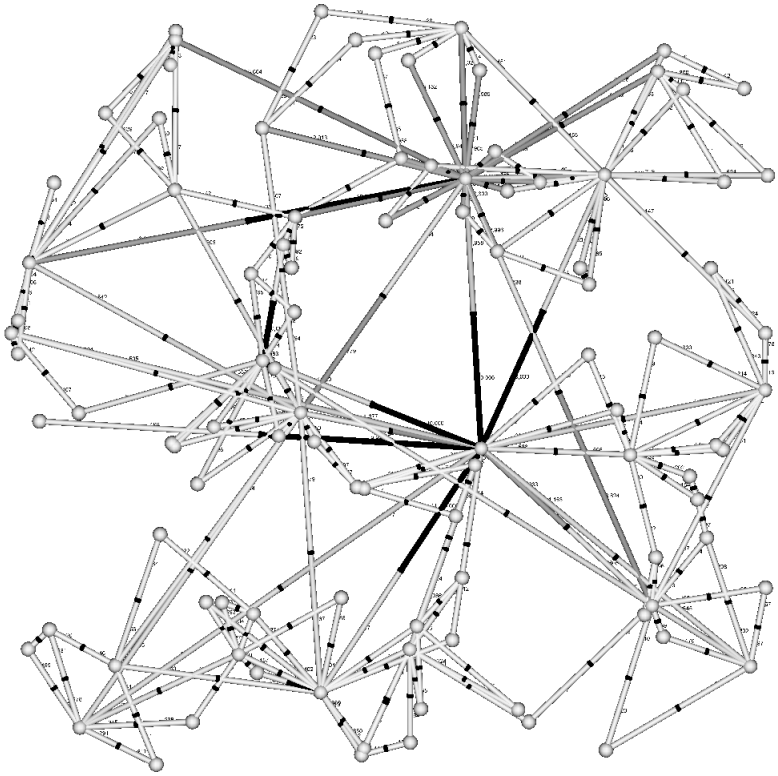

Figure 3.8: Illustration of MFIREv3 DDoS

performed a scan, or otherwise been given knowledge of the current network. The current implementation assumes this knowledge is available to the worm a priori. Figure 3.9 shows the effect of the worm process in the MASON environment. Algorithm 5 illustrates the implementation of the replication worm in MFIREv3.

In a typical worm attack scenario, the attack surface is initialized by setting up several active vulnerabilities in the environment. Next, InsecureProcesses are set up at every Node. Each InsecureProcess is initialized with a random subset of the active vulnerabilities. The InsecureProcesses are set to listen on a small number of ports. This is often the case in reality, where vulnerabilities are typically associated with specific applications, and these applications often run on a single well-known port.

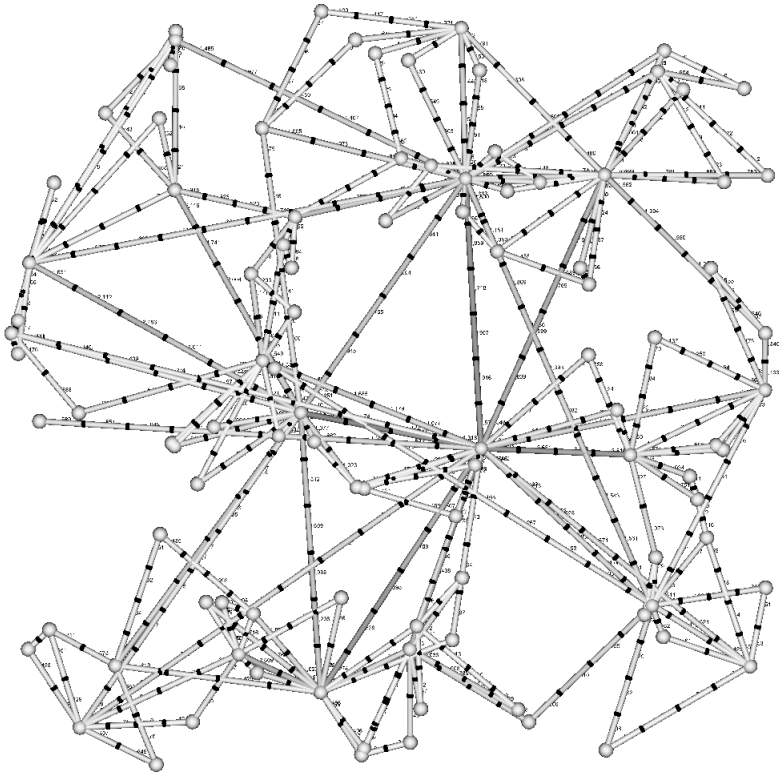

Figure 3.9: Illustration of MFIREv3 Worm

Algorithm 5 Worm Attack Implementation

```
Select target node addresses
Instantiate possible vulnerabilities
List target ports
Select rate of attack (how many attack packets to send each time step)
Set up InsecureProcesses on subset of Nodes (vulnerabilities and ports)
Time parameter before worm spreads
for Worm Active (When time parameter elapses to end time) do
    Send attack packets determined by rate to selected address and port
    if MaliciousCode execute() is called then
        Malicious payload delivered
        Run WormInstaller on new HostNode
    end if
end for
```

When the worm becomes active, on each timestep it sends as many attack packets as its rate allows. Each packet is sent to a randomly selected address and port within the initialization parameters. The packet is crafted to simulate exploitation of a randomly selected vulnerability from its arsenal.

The effect of the worm propagation is achieved when the malicious code section runs the program WormInstaller. It is a subclass of the abstract Payload class, which specifies one method that must be implemented: execute(). This method is called by the InsecureProcess if it is successfully exploited. The user can define new Payloads to run various worm attacks. For the WormProcess, the Payload is a WormInstaller. It has the sole purpose of installing a WormProcess on the host Node.

When an InsecureProcess receives an attack packet, it determines whether the active vulnerability is one to which it is exposed. If so, the exploit is successful with a certain probability. The probability is pulled from a map indexed by vulnerability number. This value is 20% to simulate the uncertainty in a real-world attack surface caused by patching.

A vulnerability scan is modeled after simple TCP-connect port sweep. Note that the current MFIRE environment does not implement the TCP protocol explicitly, however, all processes within the environment are configured to mimic the effects of TCP and provide replies to incoming packets as needed. In particular the *connect* message may be replied with a response equivalent to an *ACK*, *ICMP port unreachable*, or ignored. The scan process runs on a single random node, which sends connection requests to a random subset of N target nodes on the network at rate r packets per timestep. The scan sends a packet to all ports in the complete range of common port numbers. N and r are randomly selected prior to each simulation. Algorithm 6 shows the Scan process while Figure 3.10 shows an example block scan.

The Man-in-the-Middle attack places a new node inside the network and connects to two randomly selected, neighboring nodes. The attack takes the exact same data transfered

Algorithm 6 Scan Algorithm

Initialize range of addresses and ports
Set rate parameter (how many packets sent each timestep)
Instantiate Source nodes with DoSProcess
for *(All < destination; port > tuples)* **do**
    Send CONNECT payload
    if *UNREACHABLE is received* **then**
        < *destination*; *port* > tuple is CLOSED
        Else < *destination*; *port* > is OPEN
    end if
end for

```
[1] scanned nodes [18] - [19], ports 40 - 60

        [18] profile:
                40      closed
                41      closed
                42      closed
                43      closed
                44      closed
                45      closed
                46      closed
                47      closed
                48      closed
                49      open
                50      closed
                51      closed
                52      closed
                53      closed
                54      closed
                55      closed
                56      closed
                57      closed
                58      closed
                59      closed
                60      closed

        [19] profile:
                40      closed
                41      open/filtered
                42      closed
                43      closed
                44      closed
                45      closed
                46      closed
                47      closed
                48      closed
                49      closed
                50      closed
                51      closed
                52      closed
                53      closed
                54      open
                55      closed
                56      closed
                57      closed
                58      closed
                59      closed
                60      closed

Scan took 290 steps to complete.
```

Figure 3.10: Illustration of MFIREv3 Scan Report

from *Node A* and transfers it to *Node B* and vice versa. The extra node also holds delays data transfer speeds by holding onto the communication for a short period of time before transmitting it to the receiving node. The packets are thus delayed extra timesteps and might expire before reaching their target. Also, the packets all contain lower time-to-death values

than if they did not hop through the malicious node. The implementation and illustration are in Algorithm 7 and Figure 3.11, respectively.

Algorithm 7 MitM Implementation

```
Select target node A
Select target neighbor B
List target ports
Time parameter before transmitting packet
for All packets from Node A do
    Read < destination; port > tuple
    Hold for Time Parameter
    Pass to Node B
end for
for All packets from Node B do
    Read < destination; port > tuple
    Hold for Time Parameter
    Pass to Node A
end for
```

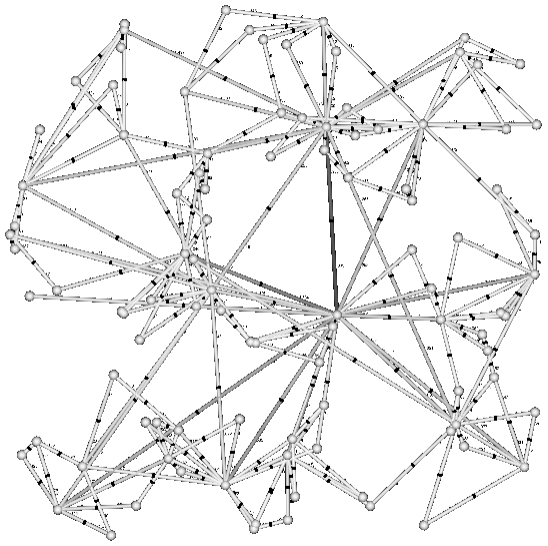

Figure 3.11: Illustration of MFIREv3 Man-in-the-Middle Attack

The Trojan attack, attacks a randomly selected node and makes frequent calls to the network hub to send traffic outbound from the simulated network. This simulates the theft of information to an outside attacker. Besides communicating with the outside attacker, the attack slows down all traffic through the node in order to steal data. The attack also opens many ports on the node, using the InsecureProcess developed for the Worm Propagation Algorithm 5, to open as many access ports to the node as possible. This insures the attacker can re-access the node if not all ports are properly secured when the attack is discovered. The Trojan attack is illustrated below in Algorithm 8

Algorithm 8 Trojan Implementation

```
Select target node
Run InsecureProcess
Set rate for updates
Set time delay for packet theft
Time parameter before worm spreads
for Trojan Active (When time parameter elapses to end time) do
    Send updates to outbound node
    if Access port closed then
        Run InsecureProcess
        Find available open port
    end if
end for
```

As stated previously, the attacks implemented in the current research are designed primarily to test to the effectiveness of the MAS reputation system. Additional attack models should be explored in future research as well as variations of the above attacks. The number of variations of attacks is infinite, however flows, trends, and statistical variations provide good statistical generalization for many of these variations.

3.3 Training the Agents

With the simulation environment completed, a priori agent training is required in preparation for executing experiments. Training the agents is vital in providing the tools

to the agents in order for the to classify attacks as described in Section 2.2. Agent training consists of generating the training data, followed by training the classifier on that data. In general, we refer to generating training data as running in *offline* mode, and testing the MAS as running in *online* mode. Many of MFIREv3's functionality performs the same in both modes, and Figure 3.12 shows the relationship between the two. In offline mode, agents do not make classifications or move; they are merely located in the network to observe and log flow-based traffic statistics. A classification model and scale file are the outputs generated from this data. In online mode, agents are making active classifications and moving in the network. The agents' classifiers use the classification model and scale file as inputs.

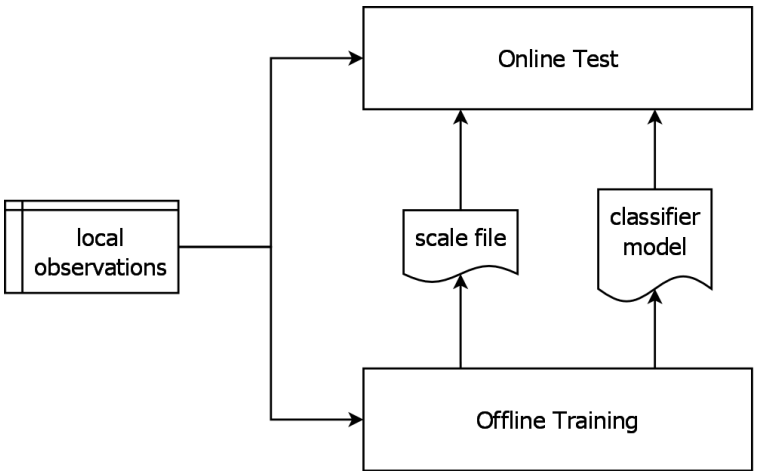

Figure 3.12: MFIREv3 offline training and online testing execution paths

The entire training and testing process is described in three high-level steps:

- 1. generate training data (MFIRE offline mode)
- 2. train the classifier (external process)
- 3. test the MAS (MFIRE online mode)

The first two steps are conducted once. After the classifier is trained, the same one is used in all agents, for all experimental models: reputation, free-movement, and deterministic. The final step is performed multiple times, as needed for the final experiments. Using the same classifier allows us to compare the effectiveness of all three models for accuracy, speed, and functionality. Additionally, agents must have the same classifier, even if heterogeneous features are used, to establish a standardized test to compare agent performance. Note that steps one and three take place within the MFIREv3 framework directly, and once set up do not require user interaction. Step two requires additional user interaction, and is conducted with external software packages.

3.3.1 Generating training data.

The purpose of generating training data is to establish a baseline of information that the agents can use for classification of attacks in MFIREv3. The background of this process is discussed in Sections 2.2.7 and 2.2.9. The primary software class for creating training data is contained is *DataGenerator*. This executes simulations in offline mode—that is, agents are located in fixed, random positions and do not move or generate attack classifications. In all other respects, the simulation environment behaves exactly the same way as online mode. To create training files, two agents are located in the network, and record all local flow-based statistics observed at their node. These files represent raw local data only.

After all of the required simulations are performed, several functions are applied to the local feature data. First, the two local feature files are combined to create a single combined feature file. Recall that local features and combined features are used separately, and an individual classifier is created for both. Local features from two different agents are converted into three combined features: average, multiple, and difference of the two. From this point on in the process, the local data and combined data remain distinct entities and are treated in parallel, although they are handled in the same way. Second, the feature

elements are scaled to between 0 and 1, and a scale file is created, to be used later in testing. Third, the data is split into separate training and validation sets.

An additional operation to scrub the data is added to this process as well. Scrubbing outliers has been shown to reduce training time and improve classification generalization by reducing overfitting [172]. This is part of the current research not formerly part of the MFIREv2 system.

3.3.2 Training the Classifier.

The chosen classifier for this research is a Support Vector Machine (SVM) (see [79, 100]). Classifiers are discussed throughout Section 2.2 with SVMs detailed in Section 2.2.9. SVM is selected due to its "high generalization performance without the need to add *a priori* knowledge," even in the face of many features [37]. Other classifiers present many potential alternatives (see Section 2.2), and should be examined further in future research. Note that the MFIREv3 environment is written to work with any classifier.

Although ANNs provide excellent qualities, the SVM outperforms a Neural Network in many instances [177]. Feature selection provides "optimal" features allowing better classification with an SVM. Lastly, the data flows remain relatively consistent between attacks. There is not new information presented after training that the classifier needs to learn.

To realize an SVM implementation, the LibSVM package [36] is selected due to its Java integration and its useful grid search method for finding optimal training parameters. Further details of the LibSVM package and alternative packages can be found in section 2.2.9. LibSVM provides the needed multi-class classification technique, implemented internally as the standard one-vs-one model. We make use of the LibSVM library function *svm_predict*, standalone executables *svmtrain* and *svmscale*, and python script *grid.py*.

3.3.3 Feature Selection.

Feature selection is another important aspect to training a classifier. If a smaller subset of quality features can be provided to the classifier, it is faster to train, and may improve classification generalization by reducing overfitting. One possible approach is to use Bhattacharya coefficient analysis [77]. Another simpler, albeit more computationally intensive, approach is the "leave-one-out" method. In this, the classifier is tested multiple times, each with leaving one feature out. In this way, the experimenter can see which features are useful and which are not. This method is crude in that it treats features as singular entities and does not consider the combinatorial effects they may have. A third method is to do a search (see Section 2.6.2) for useful features. The search algorithm is the primary method for feature selection used in this research effort, however on a number of occasions the Bhattacharya coefficient is utilized for feature selection between two classes and two features for quick comparison.

The initial simple algorithm for feature selection is shown below in Algorithm 9.

Algorithm 9 Embedded Genetic Algorithm

Initialize :Randomly generate an initial population of feature subsets encoded in binary strings of all features.
while *(not converged or computational budget is not exausted)* **do**
1. Evaluate fitness of all feature subsets in the population based on $J(s)$.
2. Select the elite chromosome s_i to undergo filter method based on local search.
3. Replace s_i with an improved new chromosome s_i' using Lamarckian learning.
4. Perform evolutionary operators: restrictive selection, mutation, and crossover.

end while

For a given candidate solution S encoded in a chromosome, X and Y define the sets of selected and excluded features encoded in that particular chromosome S. Each iteration generates a population of 30 candidates over 80 iterations. The ADD operation selects a highly correlated feature from Y and adds it to S and the DEL operator finds the least correlated feature in X and deletes it from S. This process is illustrated in Figure 3.13.

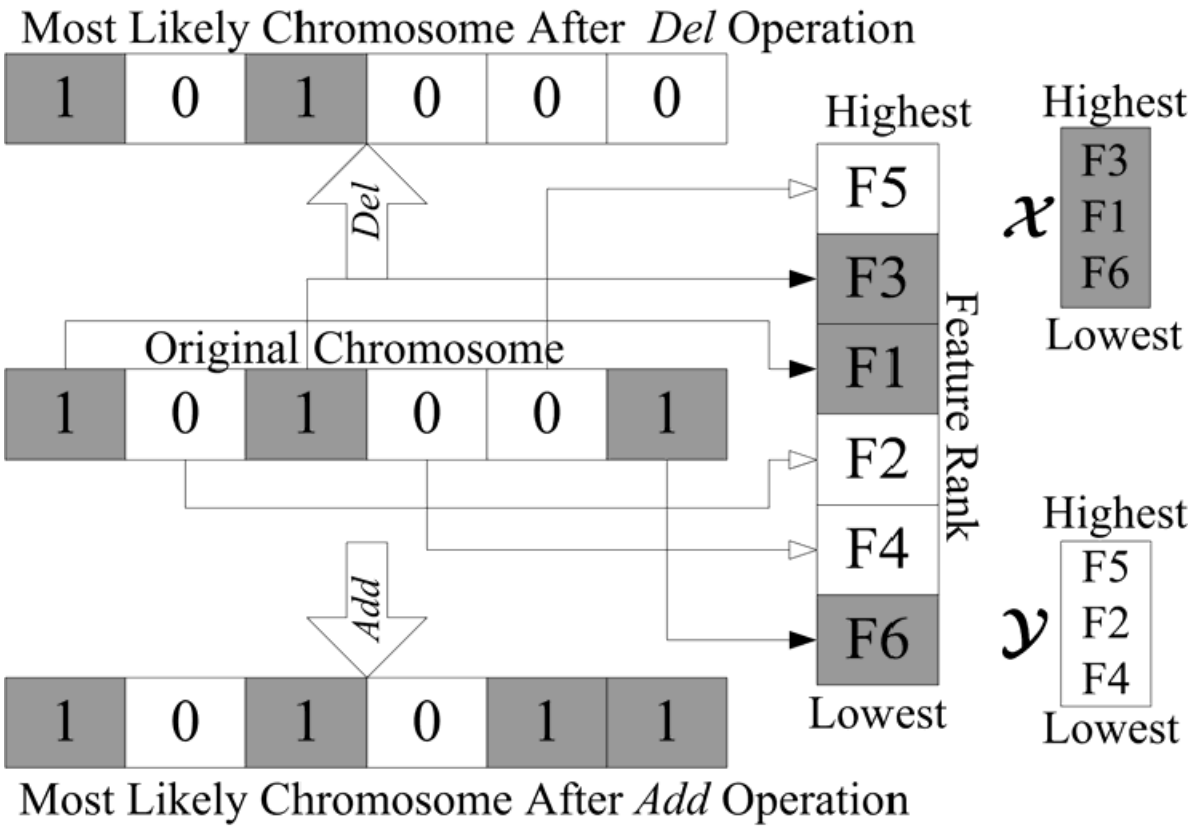

Figure 3.13: Add and Del Mutations

Algorithm 9 illustrates a single objective memetic algorithm for feature selection. It is a hybridization of a filter method based local search and a Genetic Algorithm (GA) Wrapper method. At the start of the search, the initial population is randomly initialized with each chromosome encoding a candidate feature subset. Each chromosome is composed of a bit string equal to the length of the total number of available features. In the case of the MFIREv3 design, this would be 84 features for the combined set and 24 local features for each agent. The encoding of the chromosome is simply a bit string where a '1' represents the feature is included in the subset and a '0' represents exclusion from the subset. The fitness of the chromosome subset is then determined using generalization error. In order to keep chromosomes with smaller numbers of features, if two chromosomes have

miss rates within a certain epsilon difference, the chromosome with fewer features is given a higher chance of surviving. The Lamarckian learning [71] forces the genotype to reflect the result of improvement through placing the locally improved individual back into the population to compete for reproductive opportunities.

The main shortfall of Algorithm 9 is that it does not distinguish Partial-Class Relevance (PCR) from Full-Class Relevance (FCR). PCR features are those that are only capable of differentiating between a subset of classes (attacks), while FCR features help provide distinction between all classes. For the initial part of this research we want all agents to utilize the same features, therefore FCR features provide a greater advantage. When testing agents with different feature sets in order to detect specific attack types with specific agents, PCR and FCR features are less vital and Algorithm 9 provides faster selection than Algorithm 10. Future research could introduce adaptive agents that adapt the features utilized based on the flows present in the network. For that effort, Algorithm 10 could provide the initial features for each agent. and based on a positive attack reading choose a specific feature subset from Algorithm 9 to better classify the attack.

Each iteration generates a population of 30 candidates over 80 iterations. As previously stated, the Single-Objective Memetic Algorithm did not distinguish between PCR and FCR features in multiclass problems. The true FCR and PCR features are computationally intensive to find, so in most cases they are approximated using a One-Versus-All (OVA) scheme.

The search for optimal PCR feature subsets of k OVA sets can naturally be casted as a multi-objective optimization problem with each objective corresponding to the feature selection accuracy of each OVA set. The MOP considered is thus defined as:

$$min\ F(s) = (fj(s), ..., fk(s))\ subject\ to\ s \in S \tag{3.1}$$

Algorithm 10 Embedded Multiobjective Memetic Algorithm

1. t = 0
2. Initialize :Randomly generate an initial population $P(t)$ of feature subsets encoded with binary strings.
3. Evaluate fitness $F(s)$ of each solution in $P(t)$.
4. Rank $P(t)$ using Pareto dominance and calculate crowding distance.

while *(Termination Criterion not Fulfilled)* **do**

5. Select a temporary population $P'(t)$ from $P(t)$ based on Pareto ranking and crowding distance.
6. Perform crossover and mutation on $P'(t)$.
7. Evaluate fitness $F(s)$ of each solution in $P'(t)$.
8. Rank $P'(t) \bigcup P(t)$ using Pareto dominance.
9. Apply filter method based on local search on the non-dominated solutions of 8, and generate an improved population $P''(t)$
10. Rank $P(t) \bigcup P'(t) \bigcup P''(t)$ using Pareto dominance calculate the crowding distance.
11. Select solutions from 10 to create a new population $P(t+1)$ based on Pareto dominance and crowding distance.
12. $t = t + 1$

end while

$$fi(s) = -Acc(s, ci, c'i), \ (i \in (j, ..., k)) \tag{3.2}$$

where $F(s)$ is the objective vector, s is the candidate selected feature subset, k is the number of classes, and S is the feasible domain of s.

Once again in Algorithm 10, the start of the search begins with a randomly generated list of initial population solutions with each chromosome encoding a candidate feature subset. Each chromosome is composed of the same type of bit string for inclusion and exclusion of a feature within a candidate feature subset. In each generation the offspring population $P(t)$ is generated from mutating the parent population $P(t)$. Then a non-dominated sorting categorizes the solutions of the mating pool into levels of Pareto Fronts and the non-dominated solutions are filtered using the ADD and DEL operations shown in Figure 3.13. Elitism and Diversity is maintained based on Pareto Dominance and Crowding

Distance. The evolutionary operators applied in this algorithm are binary tournament selection, uniform crossover, and mutation operators.

Following the implementation of the One-Versus-All scheme, a Partial-Class Relevant subset is created for each class. There is also an optimal Full-Class Relevant subset that is created, and assuming that the maximum number of features is not reached by the FCR subset there must be a method for combining additional features from the PCR subsets. In the current state of the algorithm only the FCR subset of features is used, thus eliminating as many features as possible. The next step would be to employ a method of adding features based on the number of PCRs they are in and breaking ties in a round robin fashion. The most common method besides keeping all features is to employ an ensemble scheme where each of the $k + 1$ feature subsets is employed for classifying all classes and the predictions of all trained classifiers are then aggregated based on voting.

All three feature selection processes; Bhattacharya, Algorithm 9, and Algorithm 10 are implemented in [51].

3.3.4 Kernel Method Selection.

This section focuses on the potential for applying varying Kernel Methods into Multi-Agent Systems as discussed in section 2.2. Using different Kernel functions as opposed to the commonly employed "affinity functions" for MASs allows the classifier to operate in feature space as was done in previous iterations of MFIRE [77, 185].

Kernel methods have been extensively studied in pattern recognition and machine learning over the last decade, and they have been successfully utilized in a variety of applications [38, 170]. A main advantage of kernel methods is that nonlinear problems such as classification and regression can be solved using classical linear approaches. This is essential for MFIREv3, as the five attacks and normal dataflow from our data sets are not linearly separable as illustrated in Figure 3.14.

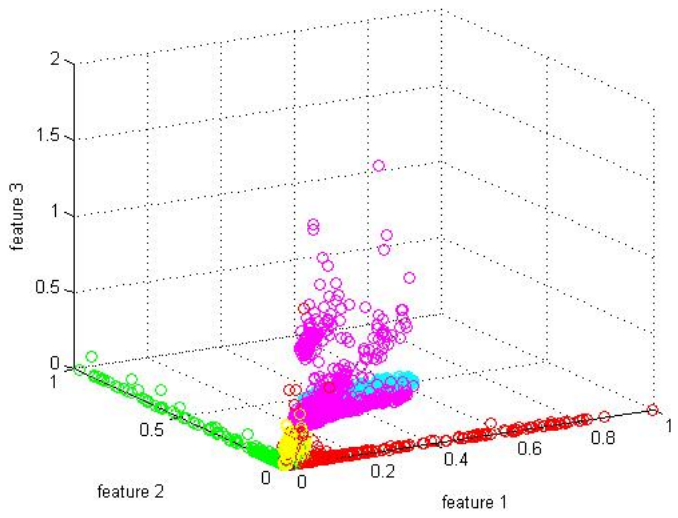

Figure 3.14: Illustration of Non-Linear Separability: Magenta-DDoS Worm- Red Scan-Green MitM-Yellow Trojan-Blue Normal-Black(Covered); Feature 1: Number of distinct destination addresses Feature 2: Total number of inbound bytes Feature 3: Std. Dev-Ratio of packets to (dest addr, dest port)

Kernel machines have been shown to outperform many other techniques in regression and classification problem, however its performance is highly dependent upon the kernel function and hyper-parameters used [38, 170]. Unfortunately, there is no analytic method to help the user discover the optimal kernel function or hyper-parameters. This means that the common approach is a simple "trial-and-error" methodology that severely limits the range of kernel functions that can be considered. Arjan Gijberts [67] presents an automated approach for finding good kernel functions and hyperplanes using Evolutionary computation, however this effort also uses the "trial-and-error" approach.

Previous versions of MFIRE utilized a Radial Basis Function (RBF) kernel for classification also known as a Gaussian kernel [185]. Using a Gaussian Kernel allowed for excellent classification accuracy. Gaussian kernels are among the most widely used and researched kernels in the field, however they are much less computationally efficient than

linear or uniform kernels [153]. A selection of common Kernels employed by SVMs are illustrated in 3.15.

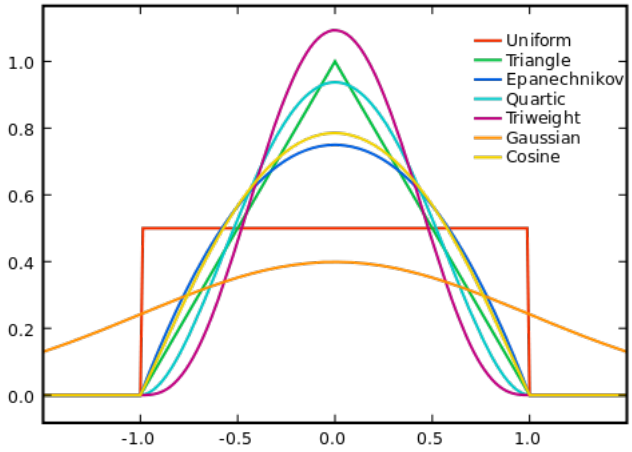

Figure 3.15: Kernels in Common Coordinate System [77]

A Gaussian kernel is of far greater complexity than a linear kernel, with a complexity of $O(nd)$ as opposed to $O(n + d)$. To begin the process of improving classification rates without decreasing accuracy drastically, we aim to find a linear kernel classifier. By using a linear classifier, parallel SVM, and fully distributed agents, the rate at which the training and classification processes complete should greatly improve from the previous version.

This effort compares the accuracy and time of classifying attacks using both a Gaussian and linear kernel. The linear kernel certainly provides faster classification. This effort compares the loss in classification accuracy as compared to the Gaussian kernel.

3.3.5 Testing the MAS.

During online testing, agents are instantiated in the environment, observe and share feature information, and provide attack classifications. The class *OnlineTest* is the primary method for performing online testing of the MAS performance.

3.4 Movement models

The next primary element of MFIRE is the functionality which controls agent movement. Recall that the *goal* of this research is to develop a Multi Agent System (MAS) for the defense of flow based system attacks and anomaly detection and identification. The following high-level *objectives* support this goal:

- Continue design and evaluate a multi-agent intrusion detection system using a Reputation system
- Evaluate the MFIREv2 multi-agent intrusion detection system using stochastic search
- Design and evaluate a multi-agent intrusion detection system using deterministic search with search incentives and Maximum Cover

In this iteration of the research development, the agents use the same classifier for both of the stated objectives. We are interested in comparing the performance of the MAS using two different models for the way agents move in the network. The following sections describe the reputation model and the free-movement model. These two models are compared to a baseline model of non-moving agents (fixed model). The fixed model is trivial, and is not described here in detail.

3.4.1 Agents using a reputation model.

The collective activity of the population of agents is tied together at the multi agent system (MAS) level through a series of providers, which process the classification decisions ('votes') of individual agents and reports the majority result amongst the rest of the providers. The reputation framework is discussed in section 2.8 as well as alternative methods to the reputation calculation used in MFIREv3. Previous versions of MFIRE chose to allow a single *AgentController* determine movements and overall network classifications [77, 185].

In MFIREv3, prior to classification agents may receive sharing assignments from their provider, and share feature values accordingly. Each agent is then able to make a classification based on local as well as shared feature values. This is a simple evolution from the Reputation system utilized in the previous versions of MFIRE [77, 185] by distributing the intelligence of *one* AgentController amongst multiple agents. The rational for moving to a distributed system is discussed in the distributed agent section 2.7.

Algorithm 11 Reputation Calculation

denote classification by agent a_j at time t using only local feature values as l_{jt}
denote classification by agent a_j at time t using combined local and shared feature values (e.g. from peer agent a_i) as c_{jt}
denote the majority classification at time t as m_t

Require: $0 < decay \leq 1$

```
procedure CalculateReputations(decay)
    for all agents a_i do
        for all recipients a_j of information provided by a_i do
            if c_jt = m_t then
                if l_jt = m_t then
                    rating_ij ← neutral
                else
                    rating_ij ← positive
                end if
            else
                if l_jt = m_t then
                    rating_ij ← bignegative
                else
                    rating_ij ← smallnegative
                end if
            end if
            reputation_i ← reputation_i + rating_ij
        end for
        reputation_i ← reputation_i × decay
    end for
end procedure
```

The provider of an agent stores the agent reputation. Each round, it calculates a rating for each sharing assignment an agent was given. The rating depends on a heuristic measure of how much the shared feature values helped or hurt the recipient's ability to classify in step with the majority. After all ratings are processed, the provider may then decay each agent's reputation by 10%. The idea is to motivate agents to explore other nodes when they are not perceived as making any positive contributions to the community.

Algorithm 11 details the idea. The values for variables *neutral*, *positive*, *bignegative*, and *smallnegative* are reflected in Table 3.2.

MFIREv3 employs a distributed reputation system per the broad categorization of [88]. This is different from the previous generations of MFIRE which utilized a central controller. This approach puts the reputation of each agent under the control of a specific provider (another agent) as opposed to a central reputation manager. As discussed in section 2.7, distributed intelligence provides many benefits in the management of complex problems. Distributing the intelligence promotes a more thorough decomposition of the complex problem. Fully distributed systems create complex management of agent interactions, but is not prone to single point-of-failure issues. Both systems offer unique benefits, and it is possible that the centralized approach is a better more sustainable architecture, however utilizing a distributed approach allows for thorough quantitative comparisons between the two architectures.

Agents start with a base reputation value of 0.5 ± 0.05, which is approximately twice the migration threshold value of 0.25 used in experimentation. The *Provider* uses Table 3.2 to modify reputations according to how well agents' observations helped receivers vote in step with the majority. When an agent moves to a new node, its reputation is reset to the base value. The small ±0.05 random offset imparts some non-deterministic movement into the agents, and is needed to combat an observed behavior which causes agents to move in lockstep with each other. This occurs if agents all make the same classifications period

Table 3.2: How the Provider Rates Shared Feature Values

Receiver's classification result, based on feature sets used		
Local Only	**Local + Shared**	**Rating**
Same as majority	Same as majority	+0
Same as majority	Differed from majority	-0.1
Differed from majority	Same as majority	+0.1
Differed from majority	Differed from majority	-0.05

after period, which happens when they are using a very accurate classifier that is not very sensitive to location in the network. Moving all at once is undesirable because no attack classification is given if all agents are in motion. Recall that agents can either be classifying an attack or moving, not both. In addition to this random offset, we also only allow at most 50% of the agents to move at one time. This ensures that in every period there are enough stationary agents to make a classification.

When agents vote in step with the majority, and would have done so even without the use of the shared observations, there is no reason to rate their providers positively or negatively. On the other hand, if the agent is prepared to vote in step with the majority, but ends up not doing so due to the influence of the shared observations, the judgment of the crowd is viewed as superior to the opinion of a single peer and thus the provider rates negatively. Real benefit is perceived when they would have voted out of step with the majority but for the "corrective help" of the shared observations, and in such cases providers rate positively. If the agent votes out of step with the majority and would have done so even without the shared observations, the provider rates negatively. But, not so

much as if the shared observations had dissuaded the agent from otherwise voting in step with the majority.

3.4.2 Agents using a Stochastic Model.

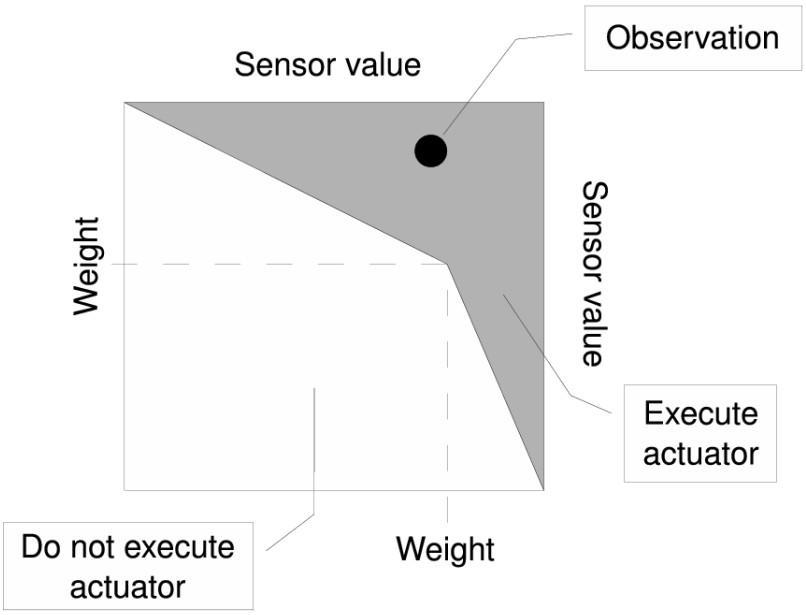

Figure 3.16: Classification Rule

In the reputation model, agents rely on a provider to provide move or stay commands. The Provider for an agent, which is another assigned agent, keeps track of reputation of each agent, which is used for both sharing assignments and movement. The agents themselves do not make any local decisions. In contrast, this model decouples agent movement from the reputation system, and allows agents to move freely on their own. The Providers still keep track of the reputation of each agent for feature sharing. Each agent controls its own movement locally, via an actuator which provides a binary activate or non-activate (boolean) decision. SOMAS [84] uses this approach and MFIREv2 [185] implemented the free-movement model to test functionality, but did not provide thorough

testing of classification accuracies or efficiency. SOMAS [84] implements the actuator shown in Figure 3.16, which takes weights from a center point to determine a binary activation area. These weights are found using a genetic algorithm.

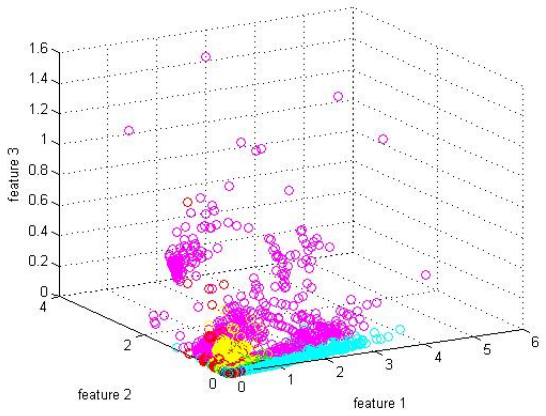

Figure 3.17: Movement Actuator 3-feature Color Determines Probability

This research implements a model designed by Wilson [185], and incorporates a stochastic element to the activation decision. It is important to keep agent movement in the network somewhat randomized, so that they do not cluster at the same node and never explore other areas. In addition, we like for a potential actuator to be easily manipulated with a stochastic search routine, such as genetic algorithms. The solution we propose is a probabilistic segmented actuator, shown in Figure 3.17. This actuator takes inputs from N features, and outputs a activation probability. Probability maps are stored directly in the actuator, and it functions as a quick lookup table, based on the location in feature space. To develop and train such an actuator, one must define three things: dimensionality (number of features), number of segments, and choose which specific features are selected. The movement actuator may use the same local and combined feature sets available to the agents' attack classifier. For the research, we examine a 3-feature classifier, with 16 total

segments. Features are selected from the optimal subset from the feature selection MOEA described in section 3.3.3.

3.4.3 Deterministic Search with Maximum Cover.

The goal for agents moving autonomously within a network is to move to unique locations that allow for the best classification of the type of attack to take place. If there is one attack and only one agent, the optimum location is simply to be located on the node being attacked in order to have the most accurate data as to the type of attack.

Real world network intrusion detection is not limited to one attack on one node. If one attack is occurring and all of the agents are located near that attack in order to best classify that attack, the rest of the network is open for attacks that go unnoticed. Leaving a majority of the network vulnerable for a second or third attack in order to use all of the agent resources to classify an initial set of attacks is not an acceptable solution to the classification problem. Instead the agents must find optimum locations for classifying attacks while still covering as many nodes on the network as possible.

This problem is the combination of two known Non-Polynomial (NP) Complete problems [8]. The first of the problems is a Constraint Satisfaction Problem [8], where given n agents and m attacks, what locations for the agents meet the constraints that every attack must have an agent within X nodes of the attack, and no two agents can be within Y distance of one another. The second NP Complete problem is a Maximum Cover problem [32], where given n agents that each cover Y nodes, what locations within the graph allow for the Maximum number of nodes to be covered while still meeting the constraints of the first problem.

The final and most important constraint is the movement incentive. If no attacks are detected and all n agents achieve a Maximum Cover, there is no incentive to search other nodes unless this action receives incentives. The incentives are provided by awarding value

to nodes that increases every time step that an agent is not within X nodes of the valued node.

The Objective Function of the problem is to find the least cost agent locations with regard to distance from the locations of attacks while achieving a Maximum Cover of unique nodes by the agents and searching through all nodes within the network.

The output domain contains the Graph G including the locations of the attacks, agents, and current node values. Also included in the output domain are three lists containing: the costs from each agent to each attack, the nodes covered by the agents and their respective totals, and the value of each node at the current time step. The problem can be broken into 6 separate structures that contain all of the objective data necessary for determining the optimum solution:

- Graph $G = n_1, n_2, , n_n$ Graph of nodes
- Agents $A = a_1, a_2, , a_n$ List of agents and their locations in the graph
- Attacks $T = t_1, t_2, , t_n$ List: Type of attack and their predicted location in the graph
- List $D = d_1, d_2, , d_n$ Distance between each agent and each attack (minimize)
- List $C = c_1, c_2, , c_n$ Number of unique nodes covered by agents and their locations (maximize)
- List $V = v_1, v_2, , v_n$ Values at nodes with agents on them at current time step (maximize)

Agents move in order to complete three objectives. The most vital and primary objective of each agent is to make sure a detected attack is classified by moving toward an optimal location for classification. Reputation does not exist within this movement model. Instead, all agents are assumed to maintain "perfect" reputations. If one agent detects an attack, the network is classified as being attacked by that attack and agents

move to optimally classify that attack. If two agents detect different attacks, the agents move to optimally classify both attacks and the network is classified as being under both attacks. This methodology is not optimal and leads to many false positive classifications. The objective of this model is to test the effectiveness of distributing knowledge amongst the nodes for possible inclusion in a Reputational or other movement model.

The next objective is to achieve a Maximum Cover of the network to allow for the greatest chance to detect a new attack early. The last objective is to search the entirety of the network to promote detection of attacks on exterior nodes.

3.5 Defensive Measures Methodology

Section 2.9 presented a number of defensive measures to bolster an Intrusion Detection System (IDS). Of the defensive methods presented, certain methods present complicated ethical dilemmas while others are not suitable for an anomaly based IDS. We selected two methods of defense for inclusion in MFIREv3: rate-limiting and node elimination/rerouting..

Section 2.9 illustrates the effectiveness of rate limiting in real world applications. For this research effort we qualitatively evaluate the spread of a DDoS attack with and without rate limiting. The Pareto model of normal traffic described in Section 3.2.3 limits traffic to a normalized rate below 0.1. The DDoS attack spreads at rates exceeding the normalized 0.1 threshold. Limiting the spread of traffic when the attack takes place reduces the ability of the attack to spread. This is because many attack packets must drop as normal traffic corrupts the flow of malicious packets on a chocked network. For this research experiment, we limit the traffic to a maximum normalized value of 0.1 to prevent the spread of the DDoS attack.

The elimination of a network node that is corrupted with a Worm attack can block the attack in a small area of the network [39]. Incapable of infiltrating the remaining areas of the network, a Worm attack is neutralized in the amount of time it takes to reimage the

compromised nodes. For this experiment, we block all traffic forwarded from one node in the network in order to contain the attack. The network used in this experiment is specific to testing the capabilities of containing a Worm attack and shown in Figure 3.18.

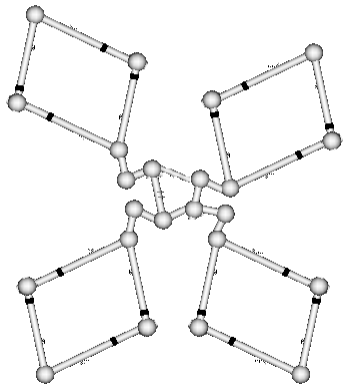

Figure 3.18: Worm Rerouting Network MFIREv3

Four agents placed at the four nodes connecting the *leaves* of the network to the center of the network attempt to classify a Worm propagation. When the attack is detected, the agent detecting the attack closes all ports on the node effectively removing the node from the network. The Worm propagation begins at one of the four corners of the network and aims to spread to as many nodes as possible before containment. At the end of each simulation, the number of compromised nodes is counted (minimum of 3). If the attack escaped the quadrant of the network it began in the defense is considered a failure. In a larger network once a Worm takes over a central part of the network it is very difficult to contain [39].

3.6 Visualization

The purpose of visualization for any exercise is to provide vivid insight into a complex system in order to demonstrate a better understanding of the system [60]. Chapters 2 and 3 present complex attack scenarios, agent movement, and network architecture that demand visualization in order to best comprehend and evaluate the research experiment and design. In many cases however, overly complex and intricate visualization tools designed with compelling animations can distract the audience from the information at hand.

The intuition behind animation is clear: if a two-dimensional image is good on paper, then a moving animation should be better [60]. This is the case for MFIREv3 as well. Quality, detailed information about the success of failure of agent movement, defense, and attack spread can be derived from two-dimensional images of the network. An improvement of this process is to visualize the agent movement up to the point where they reach optimal locations for classification.

Visualization animation helps a viewer work through the logic behind an idea by showing the intermediate steps and transitions. It offers a fresh perspective and invites the user to look deeper into the data presented.

The visualization of these movements does not need to be distractingly complex. Simple designs of networks are illustrated in the MASON package and allow for the viewer to see the necessary components of the network with ease. Agents can be represented in any number of ways, however the intuitive design is to simply represent and agent located on a node as a different color at that specific node location. Attacked nodes are represented in yet another color, with the *malicious* traffic they create represented in a different color from normal network traffic. An example from the MFIREv3 visualization system is illustrated in Figure 3.19, with malicious traffic represented in a dark gray and agents in position to classify the attack shown as dark ovals inside network nodes.

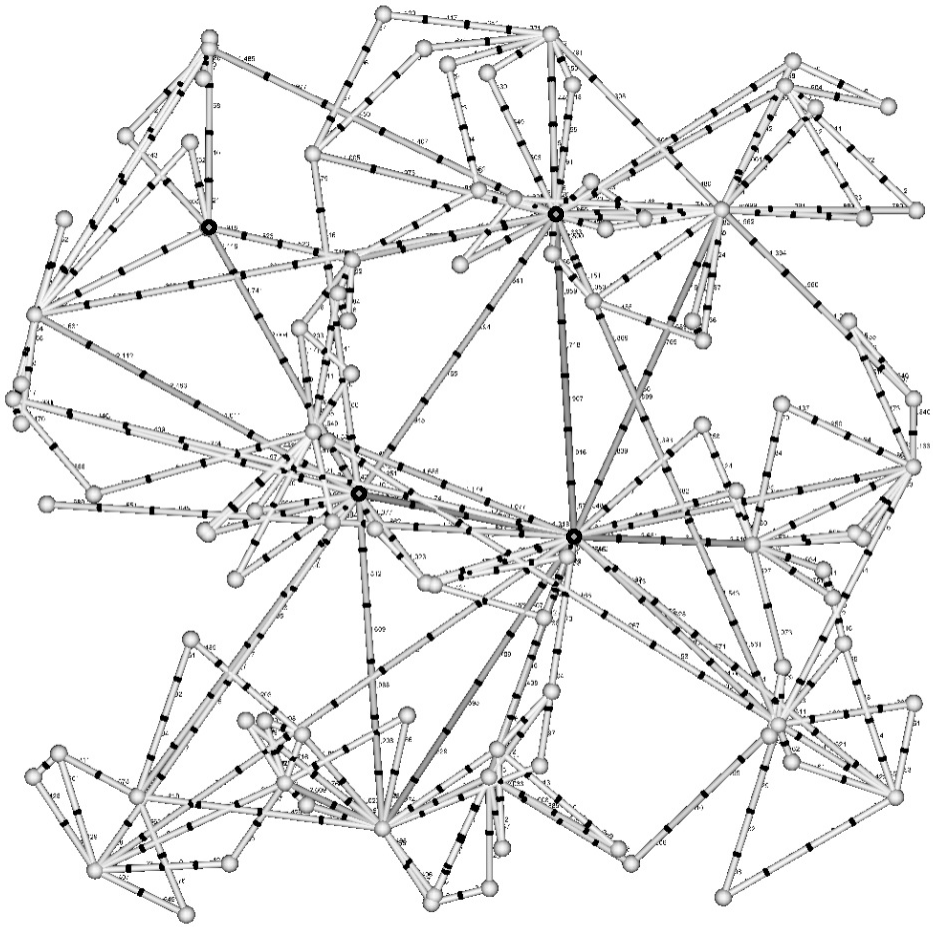

Figure 3.19: Illustration of MFIREv3 Visual Interface

Visualization however appears to fall short in representing complex data in the form of scatter-plots. Two separate studies have looked at different types of animations concerning data and found that they rarely help improve a student's understanding of the system [60].

Instead, detailed static graphs that utilize varying color schemes and angles appear to help understanding data most effectively. People have difficulty tracking more than four moving points at a time, and in the case of MFIREv3, there would be six moving points for a 3-dimensional graph in feature space.

Research shows that illustrating one concept or theory in a static graph is the best method to convey meaning [117]. For this research, outside of the animation of agent

and attack movement, data must be analyzed using static graphs. Illustrations of class separation, classification accuracy, and results of attacks does not only help convey understanding of the success or shortcomings of the system, but also helps convey the meaning of the algorithm and data themselves. The key is to not only provide the histograms, scatter-plots, or graph of the network, but to also place the images in a logical order with thorough descriptions as to tell a story that conveys the meaning in the simplest form [117].

3.7 Summary

This chapter illustrates the design and implementation of MFIREv3, including the MASON discrete event simulator, network features, feature selection, attack models, defensive measures, and SVM classification. The following chapters build upon these design implementations through experimentation and analysis of the results.

IV. MFIREv3 Experimentation and Analysis of Results

The previous chapter illustrates the design and implementation of MFIREv3, including the MASON discrete event simulator, network features, feature selection, attack models, defensive measures, and SVM classification. This chapter presents the experimentation and analysis plan evaluating the objectives stated in Section 1.2. Section 4.1 provides insight into testing a Software System. Section 4.5 describes the experimental design. Results and analysis are presented in section 4.6. The Defense experimental design and analysis is presented in Section 4.7.

4.1 Software Testing

This chapter discusses the design and testing methods and techniques for the MFIREv3 system. This section details the multiple agent testing.

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. For this effort we test synthetic data, which according to McGraw-Hill Dictionary of Scientific and Technical Terms includes "any production data applicable to a given situation that are not directly obtained by direct measurement" [140]. Instead of using real world data sets of intrusions, which contain significantly greater variation, increased attack inactivity, and lack of baseline for any class, we create a model of the network traffic that can be tested more effectively [38]. Another issue with using real network attack traffic is that much of the useful datasets are proprietary or classified and testing attacks on a live network creates fetching legality issues. Synthetic data helps meet the needs of the investigation, by modeling real attack traffic as microflows [30].

Barr et al. [18] explain the need for an experimental design that helps determine whether a new heuristic method contributes something important. They present a list of possibilities. A heuristic method makes a contribution if it is:

- Fast: produces high-quality solutions quicker than other approaches;
- Accurate: identifies higher-quality solutions than other approaches;
- Robust: less sensitive to differences in problem characteristics, data quality, and tuning parameters than other approaches;
- Simple: easy to implement;
- High-impact: solves a new or important problem faster and more accurately than other approaches;
- Generalizeable: having application to a broad range of problems;
- Innovative: new and creative in its own right.

Barr furthermore asserts [18] that research reports about heuristics are valuable if they are:

- Revealing: offering insight into general heuristic design or the problem structure by establishing the reasons for an algorithm's performance and explaining its behvaior;
- Theoretical: providing theoretical insights, such as bounds on solution quality

From Section 1.2, the *goal* of this research is to develop a Multi Agent System (MAS) for the defense of flow based system attacks and anomaly detection and identification. The *objectives* supporting this goal are:

- Continue design and evaluate a multi-agent intrusion detection system using a Reputation system

- Evaluate the MFIREv2 multi-agent intrusion detection system using stochastic search
- Design and evaluate a multi-agent intrusion detection system using deterministic search with search incentives and Maximum Cover
- Design and evaluate a Multi Objective Evolutionary Algorithm for best subset feature selection
- Determine if attacks can be classified using a Linear Kernel as opposed to the Radial Basis Function when using MOEA selected features
- Create a robust distributed simulation framework for evolving self organizing multi-agent systems
- Create a robust simulation framework for the automated defense of the network

Therefore, the reputation, stochastic, and deterministic systems under a self-organized and fully-distributed framework are the initial heuristics under study, while the feature selection algorithm and defensive measures are evaluated for improvements to the overall effectiveness of the system. Qualitatively, we can observe that these features are all innovative and effective. Our experimentation aims to demonstrate is that the use of these features increases the accuracy of the multi agent network attack classifier and improves its overall ability to identify and eliminate attacks.

The order of these objectives suggests a natural chronological sequence of testing. Feature selection testing preceded the movement model development for use in agent classification. The Reputation model developed in MFIREv1 [77] is the first model tested using the fully distributed system. This preceded testing on MFIREv2's Evolutionary algorithm [185], and testing on the new Maximum Cover Deterministic Search. Implementation of the defensive measures is the final stage of this design.

Software testing provides an objective, independent view of the software to display the risks, failures, and implementation characteristics of the product. Testing insures that the errors are eliminated or accounted for, and that in a *black-box* scenario the software is capable of providing good solutions [38].

The general focus of software testing is validating and verifying that a computer application [145]:

- meets the requirements
- works as expected
- the test can be replicated
- satisfies the needs of the user

Different software development models focus the test effort at different points in the development process [145]. Newer development models, such as Agile [112], often employ test-driven development and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed [135].

Testing software agents and Multi-Agent Systems (MAS) requires subtle differences in its methodology. For instance, the autonomous behavior of the agents as well as their distribution, social and deliberative properties, which are particular for MAS, require evaluation separate from the performance of the classification accuracy, defensive tactics, feature quality or any other quantitative measure [44, 65, 94, 156]. In other words, the agent behavior requires some partially qualitative evaluation outside of quantitative measures for the rest of the software.

Agents operate asynchronously and in parallel making testing and debugging challenging [134]. Agents communicate primarily through message passing instead of

method invocation, so traditional testing approaches are not directly applicable. Agents are autonomous and cooperate with other agents, so they may run correctly by themselves but incorrectly as a community or vice versa. Adding to the difficulties, agents in MFIREv3 are programmed to learn thus changing their behavior over time. This means that successive tests with the same data may yield different (albeit improved) results [185].

Initial works on evaluating MAS quality focused on the definition of techniques for automating the validation of MAS specifications through formal proofing or model-checking [30, 112], and on the development of debugging tools and techniques to enhance MAS development platforms [145].

Structured testing approaches have been proposed more recently, to complement analysis and design methodologies [38, 68, 135, 145]. These approaches rest on the idea that behavior of the MAS can be dynamically evaluated providing as input a set of test cases that are derived from analysis and design artifacts.

Differently from these techniques, simulation-based approaches aim at detecting abnormal behaviors while a simplified version of the system is executed in a virtual environment [44, 65, 94, 156]. This is the methodology utilized in MFIREv3 testing, because they are particularly appropriate to evaluate emerging behaviors in self-organizing systems [47]. Data mining techniques are applied to simulation logs for the small and large versions of the MAS by simply taking the output data. In other systems, simulation logs are dealt with using ACLAnalyser in order to deal with scalability issues in a large MAS [152]. This is not utilized in the current installment of MFIRE, but should be looked at for future, larger versions of the system.

4.2 K-Fold Cross-validation

In all research it is important to validate the results of the experiment by observing the results of the test data. Results of the training data show very little; only that the learning machine was able to model the training data. Given that there is no baseline for comparing

the effectiveness of the model (outside of the static model), the performance of the system is evaluated based on the Receiver Operating Characteristic (ROC) curve [134] and tables illustrating the percentages of correct classifications by the machine after training to the model. For this effort, a 5-fold Cross-validation was used over the approximately twenty-thousand data points produced.

This means that approximately twenty-four thousand points helped in the training process, while six thousand validated the results of the model. For validation purposes, it is normal to choose a value of ten for K [134], however for this effort a 5-Fold validation affords plenty of training data while allowing a larger, more exhaustive set of data points to be used for validation.

4.3 Feature Selection

Feature selection effectiveness improves the overall quality of the agent's classification ability. Section 2.2.2 explains the importance of good features in classification systems, while section 3.3.3 details our design. Testing the quality of features is completed using static agents to provide standardized results.

The test with five attacks compares the accuracies between a random set of three features (as conducted in MFIREv2 [185]) and the set of three features selected by the MOEA Algorithm 10. Agents are trained using all 84 features for twenty-four thousand samples, and validated using their respective features with six-thousand points. The training involves using only two agents, while the validation uses four static agents located at predetermined locations with high visibility of a majority of nodes on the network. The Radial Basis Function (RBF) Kernel is selected for this experiment for its accurate classification results. For each MOEA, we populate the space with 30 samples over 40 iterations.

Table 4.1: Feature Selection Classification Accuracies

	Random	**MOEA**
mean	0.642	0.698
median	0.639	0.690
stddev	0.032	0.040

The feature set provides a 13.1% increase in classification accuracy for static agents over the six scenarios. These results support the design objective: Design and evaluate a Multi Objective Evolutionary Algorithm for best subset feature selection.

We also test the classification improvement between each individual attack and the non-attack mode. This test only uses two features, but two separate feature selection algorithms choose "optimal" two-feature sets to compare with the random set. The first algorithm uses the Bhattacharya coefficient to choose the "best" two-features and the second is the MOEA; Algorithm 9. This test allows us to determine the increase in performance if all agents are trained to detect the specific attack that takes place. Once again, two static agents are trained using all 84 features. This time however, only eight-thousand points are used for each test (four-thousand attack and four-thousand non-attack), and validated using their respective features with two-thousand points.

The results illustrated in Table 4.2 show an improvement in the classification accuracies when agents use the best features for classifying an individual attack. The results are more accurate than the five-attack classification, because the agents only have to distinguish between two states.

4.4 Kernel Functions

As illustrated in section 3.3.4, the data cannot be linearly separated into classes without entering a higher dimension. To enter a higher dimensional space, we must introduce a non-

Table 4.2: Feature Selection Single Attack

		Random	**BC**	**MOEA**
DDoS	mean	0.802	0.858	0.859
	median	0.803	0.860	0.861
	stddev	0.052	0.057	0.040
Scan	mean	0.772	0.778	0.828
	median	0.780	0.791	0.833
	stddev	0.032	0.033	0.039
Worm	mean	0.782	0.824	0.869
	median	0.776	0.811	0.870
	stddev	0.025	0.031	0.032
MitM	mean	0.696	0.737	0.748
	median	0.697	0.731	0.751
	stddev	0.043	0.044	0.041
Trojan	mean	0.802	0.855	0.869
	median	0.796	0.840	0.880
	stddev	0.046	0.042	0.050

linear Kernel. Testing conducted with a linear Kernel provided inaccurate classification results. Figure 3.8 shows the classification data for the optimal pair of features for the Denial of Service attack. It is clear that a simple linear classifier is incapable of providing quality separation between the attack data and the normal non-attack set.

The other attacks can be found in Appendix F, and the figures support the use of a non-linear Kernel. Section 4.3 details the accuracy of the Gaussian Kernel with all attacks and individual attacks. Testing using a linear Kernel took place with only individual attacks.

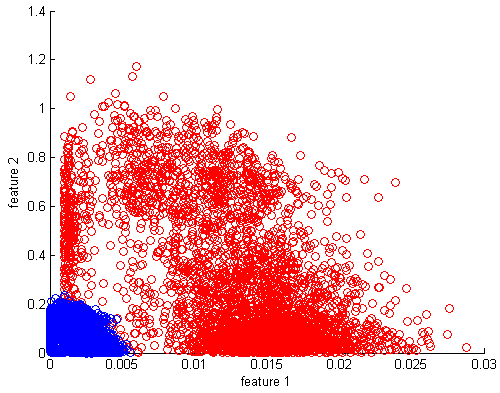

Figure 4.1: Illustration of DDoS vs Normal Features F1: Num inbound Bytes F2: Num inbound Packets

As Figure 3.8 and the Figures in Appendix F indicate, the accuracy of the classifier was substantially diminished using a linear Kernel.

The results of the linear Kernel classification using the same feature sets as the Gaussian Kernel are shown in Table 4.3.

As the results illustrate, with the current datasets a non-linear Kernel is required. In this research investigation we utilize a Gaussian Kernel, however section 3.3.4 illustrates other options.

4.5 Movement Models Experimental Design

Three movement models are designed and tested in this research effort: Reputation, Stochastic Evolutionary Algorithm, and Deterministic Max-Cover.

Testing the effectiveness of the MFIREv3 MAS using each model consists of training two classifiers (local and combined) using data generated offline, and testing the MAS online in two modes of operation. In particular, we seek to find the overall MAS classification accuracy with a 4-agent and 8-agent model. For both models, we observe the accuracy at the initial time in the simulation, and at the end. The change in accuracy

Table 4.3: Linear Kernel Accuracies

		Random	**BC**	**MOEA**
DDoS	mean	0.612	0.648	0.646
	median	0.613	0.640	0.641
	stddev	0.080	0.069	0.070
Scan	mean	0.622	0.644	0.634
	median	0.612	0.623	0.622
	stddev	0.066	0.075	0.059
Worm	mean	0.581	0.601	0.609
	median	0.589	0.601	0.600
	stddev	0.055	0.060	0.061
MitM	mean	0.598	0.612	0.619
	median	0.597	0.609	0.611
	stddev	0.062	0.059	0.058
Trojan	mean	0.571	0.600	0.608
	median	0.579	0.599	0.604
	stddev	0.068	0.068	0.069

during this time represents the increase in performance attained from agents finding better vantage points in the network.

To create the training data, we run repeated simulations on a single, 100-node regional network, where each simulation represents a single attack scenario. The attack is selected one-at-a-time from our six defined scenarios: Normal, DDoS, Worm, Scan, Man-in-the-Middle, and Trojan. Flow-based statistics are captured in two places in the network and processed to create two training sample sets, local data and combined data. We then train both classifiers with an SVM using an RBF Kernel, 5-fold cross validation, and a grid

search for optimal parameters C and γ. The two resulting classifier models are used for all subsequent testing.

To limit the variance in the experiment, all six scenarios are executed 20 times, and average accuracy recorded. We perform 30 sample observations, which yields 7200 total simulations needed. This sample size is chosen to allow first and second order statistics to be used to evaluate the results. More observations are preferred, and 30 is an acceptable number to achieve a good confidence level while still running in a reasonable time [184]. Each simulation must be performed for a minimum number of necessary time steps to ensure agents have time to move to better vantage points, and a time span of 80 time periods is conservatively allocated.

4.5.1 MFIREv3 Reputation System Experimental Design.

The Reputation Model is thoroughly tested by both the creator Hancock [77] and Wilson [185]. This research effort aims to improve upon the previous versions results with feature selection and distributed control of the agents.

Experiment summary for the Reputation model:

- Six attack scenarios: Normal, DDoS, Worm, Scan, MitM, Trojan
- Each simulation: 80 time periods
- One observation: six scenarios over 20 simulations each
- Total sample size: 30 observations
- Number of agents: four or eight

4.5.2 MFIREv3 Stochastic Search Experimental Design.

The Stochastic Search, created by Timothy Wilson [185], required full experimental testing. Wilson validated the functionality of the system in MFIREv2 [185] and this research investigation tests the accuracy of the search using the features selected in section

4.3. The effectiveness of the search is compared with the other two models and the baseline model.

The free-movement model is created by running a generational genetic algorithm on a population of candidate solutions. Each successive generation, the individuals become more adapted to solving the problem. We conduct the GA with the following parameters to validate its functionality:

- Single objective: maximize overall classification accuracy
- Individual solution: real-valued agent movement actuator
- Feature selection: three features selected
- Maximum evaluations: twenty generations

The experiment summary for Stochastic Search is defined as:

- Four and eight agents in the network
- Six scenarios: Normal, DDoS, Scan, Worm. MitM, Trojan
- One observation: six scenarios over 20 simulations each
- Total sample size: 30 observations
- Improvement: difference between final and initial accuracy
- Fitness: average improvement over 2 simulations

The test parameters and experiment are not meant as a complete method to find a near-optimal movement actuator, but provide a detailed analysis of the current effectiveness of the design. With a stochastic search genetic algorithm, infinite variations of decision parameters provide varying degrees of effectiveness in any system. Areas of optimization in the stochastic search model include:

- Fitness function
- Polynomial mutation
- Simulated Binary Crossover (SBX)
- Binary tournament selection
- Parent-child replacement
- Convert actuator to chromosome
- Convert chromosome to actuator
- Save actuator model
- Read actuator model
- Agent behavior

The stochastic search model provides an alternative movement model for the agents with different areas of optimization from the other two models. All three models can be optimized independently or specific benefits of one model can be brought into the other models.

4.5.3 MFIREv3 Deterministic Maximum Cover Algorithm Experimental Design.

This final movement model is a Deterministic Maximum Cover model, using a Multi-Objective Evolutionary algorithm. The purpose of exploring this avenue in agent location optimization is determining the validity of distributing knowledge of node quality among the nodes in the network. The deterministic model is created by running an optimization algorithm on the agents. Each successive generation, the individuals choose to search new locations, while maintaining a "good" cover of nodes. We conduct the algorithm with the following parameters to validate its functionality:

- objectives: maximize classification accuracy, coverage of network, exploration of network
- Individual solution: choose best valued location based on rewards of cover or new node
- Feature selection: three optimal features selected

The experiment of the Deterministic Search is defined as:

- Four and eight agents in the network
- Six scenarios: Normal, DDoS, Scan, Worm. MitM, Trojan
- Simulated for twenty observation periods
- Improvement: difference between final accuracy and initial accuracy
- Fitness: average improvement over 2 simulations

Once again, the test parameters and experiment are not meant as a complete method to find a near-optimal model, but rather as a method for validating the core functionality of the deterministic model. The key component of the model is the retention of information about specific nodes. This feature can be applied directly to the other movement models in any number of key areas. In this instance, the nodes retain the time between their last observation. As addressed by David Hancock [77], if nodes retain the information that a node is not beneficial for classification (does not provide a good reputation), agents would be able to avoid moving to that location as frequently. Specific areas of optimization in the deterministic search model include:

- Fitness function
- Classification accuracy

- Max Cover Reached
- Nodes searched
- Agent behavior

The final results of the Reputation model, free-movement model, and the maximum cover model are presented in the next section.

4.6 Movement Model Analysis

The next three sections provide the detailed results for the experiments defined in Section 4.5.

4.6.1 MFIREv3 Reputation System Performance Assessment.

The MFIREv3 reputation system is tested with the plan described in Section 4.5.1.

Figure 4.2 shows the number of agents that move in each time period. As agents develop better reputations with time, the number of agents moving each time period begins to settle at 2 agents. This is because the reputations of the agents approach optimal locations for classification, and only a small number of the agents need to seek better vantage points.

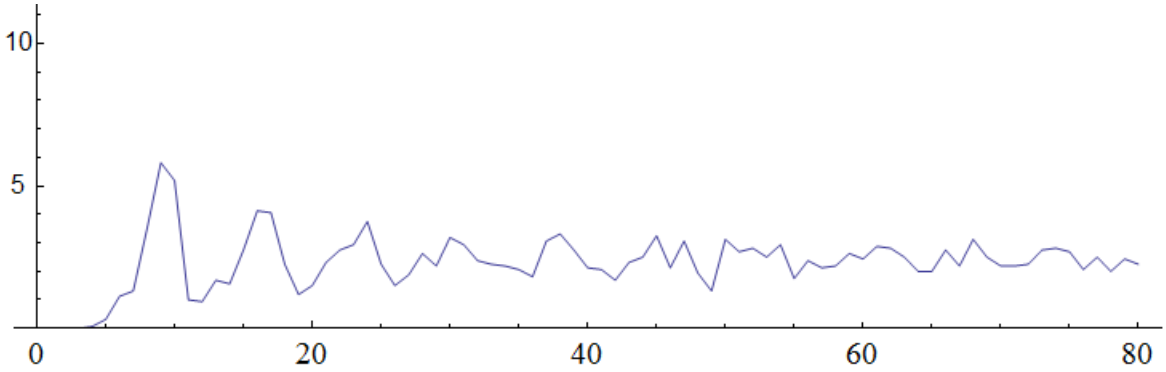

Figure 4.2: Average Agents Moving (8-agent Reputation Model)

Figures 4.3 and 4.4 illustrate the average accuracy and false positive rate for the MAS at every time period, for the 4-agent and 8-agent models. Each data point is the average of 120 simulations: 20 for each of the six scenarios.

The data shows that the false positive rate remains relatively steady throughout the simulation, however there are drastic increases in the accuracies as the agents locate improved vantage points.

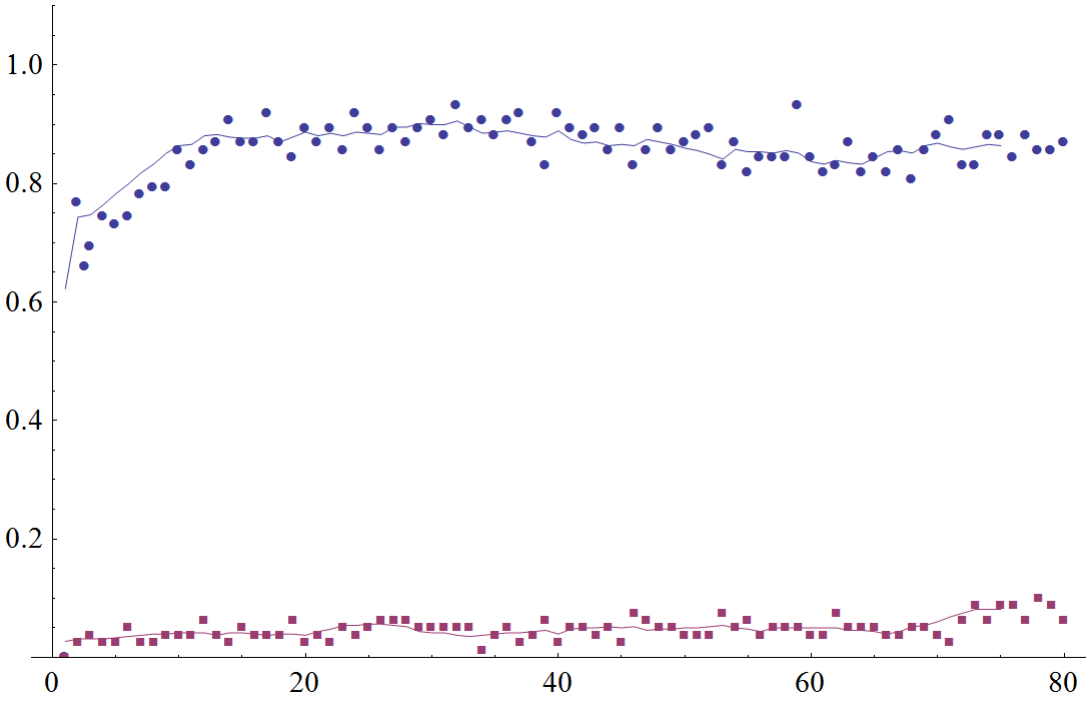

Figure 4.3: 4 Agent Reputation Model: Accuracy & False Positives vs. time

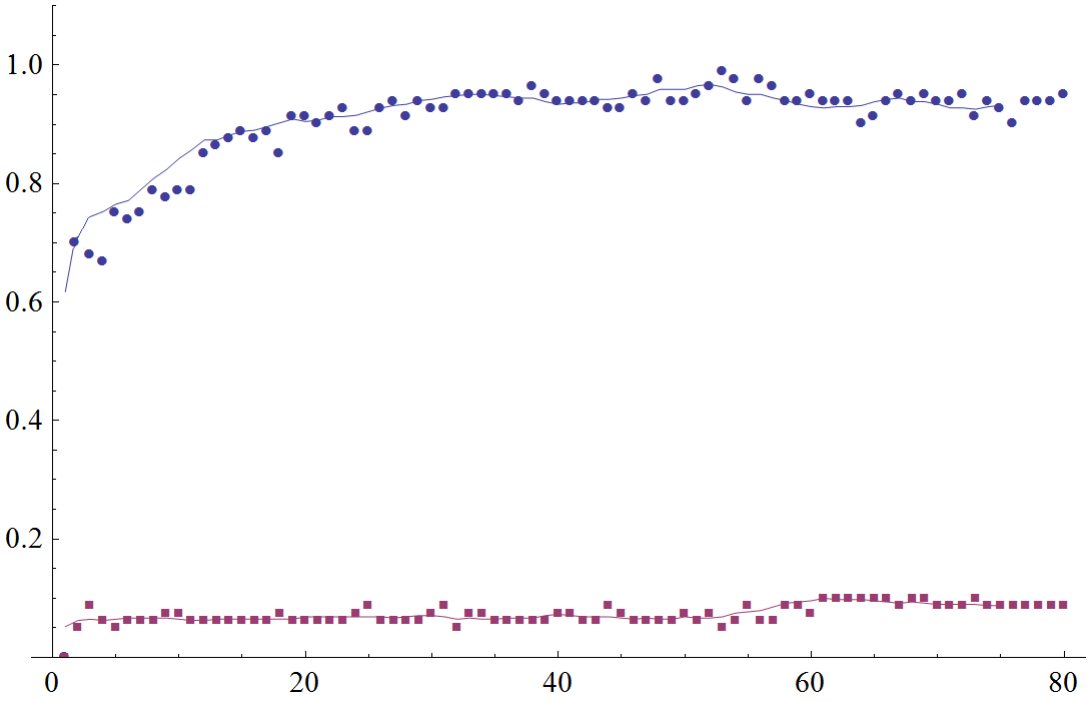

Figure 4.4: 8 Agent Reputation Model: Accuracy & False Positives vs. time

We now evaluate the relative performance of Reputation system when using 4 or 8 agents in the network. When making a classification, the result can be *correct*, a *false positive*, or a *false negative* (see Section 2.3). $Accuracy = \frac{numbercorrect}{total}$. Accuracy data for all 30 sample observations is provided in Table 4.4. The columns show results for each of the two experiments: 4-agent and 8-agent, and provides both the initial and final accuracy.

Table 4.4: Reputation System Overall Accuracy

	Initial: 4-agent	**Initial: 8-agent**	**Final: 4-agent**	**Final: 8-agent**
mean	0.703	0.698	0.891	0.940
median	0.721	0.720	0.890	0.956
stddev	0.038	0.043	0.032	0.023

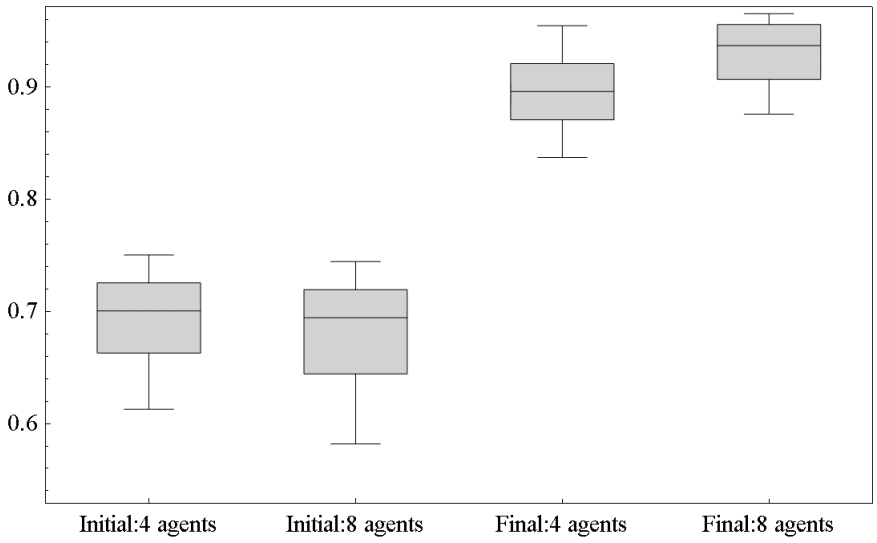

Figure 4.5: Accuracy box plots

To compare any two of the models we use Wilcoxon Rank Sum (Mann-Whitney) test [128], which is a non-parametric statistical hypothesis test for assessing whether one of two samples of independent observations tends to have larger values than the other. It remains the logical choice when the data are ordinal but not interval scaled, so that the spacing between adjacent values cannot be assumed to be constant. The Mann-Whitney test is more robust than the Student t-test, as it is less likely to spuriously indicate significance because of the presence of outliers [128]. We use the MATLAB *ranksum* function [77] to compare of all six possible combinations of the three models under test. A p-value of less than 0.05 indicates a significant difference between the two models under comparison, with 99% confidence. A p-value of larger than 0.05 indicates there is not sufficient evidence that the two models perform differently.

Table 4.5: Wilcoxon Rank Sum p-values

	p-value
Initial:4-agent vs. Final:4-agent	2.5×10^{-22}
Initial:8-agent vs. Final:8-agent	1.7×10^{-30}
Initial:4-agent vs. Initial:8-agent	4.9×10^{-4}
Final:4-agent vs. Final:8-agent	2.3×10^{-9}

The p-values between all of the models listed in 4.5 illustrate that there is significant evidence that a difference exists between the models in each category. The most important significance is the evidence that the agents improve with time in classifying attacks as time continues and the agents reach improved vantage points.

4.6.2 MFIREv3 Stochastic Search Performance Assessment.

Timothy Wilson created and successfully validated the use of a stochastic search algorithm for the MFIRE system [185]. Testing the accuracy of the system in accordance

with section 4.5.2 and the use of the "optimal" feature subset for classification allows us to compare the effectiveness of the stochastic search with the other movement models.

Figure 4.6 shows the number of agents that move in each time period. Agent movement is locally decided based on probabilities within the movement actuator. The movement actuator does not stabilize as the Reputation model does with time and agents locating good observation points. Instead, as the probabilities to move remain constant, the number of agents moving each timestep never remains sporadic.

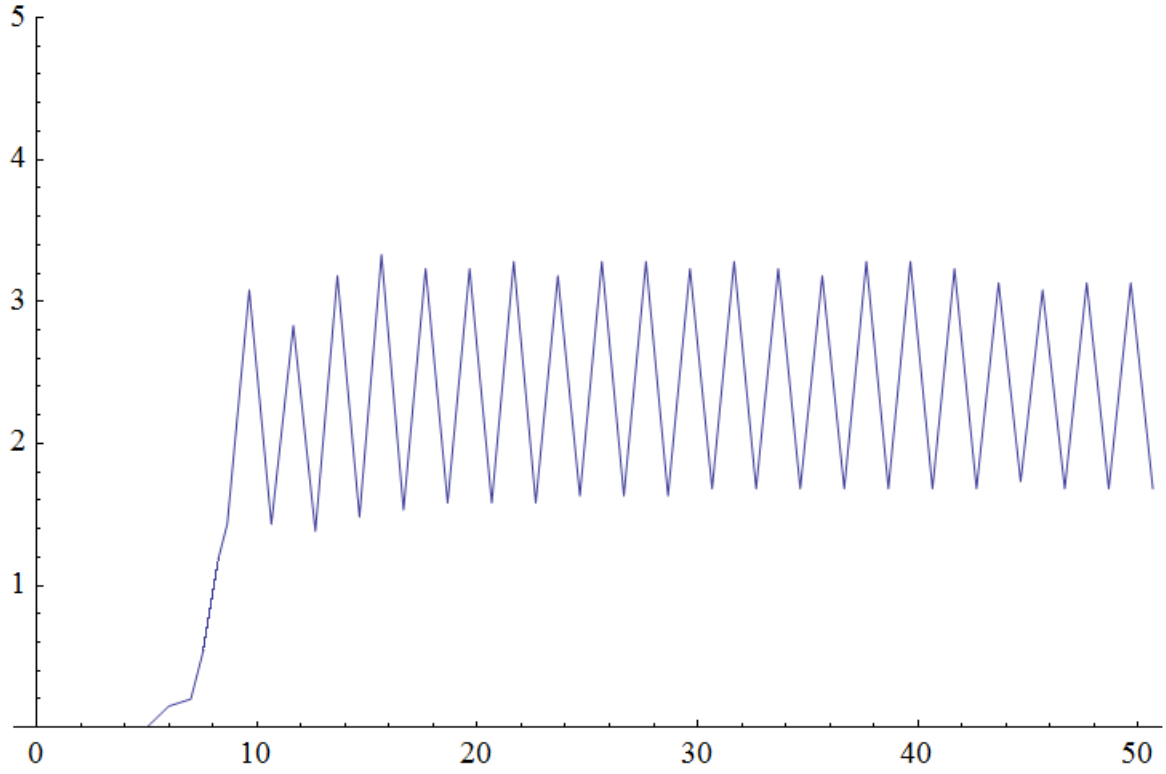

Figure 4.6: Average Agents Moving (8-agent Stochastic Model)

Figures 4.7 and 4.8 illustrate the average accuracy and false positive rate for the stochastic search model at every time period, for the 4-agent and 8-agent models. Once again, each data point is the average of 120 simulations: 20 for each of the six scenarios.

The number of false positive classifications continues to remain low, while the classification accuracy increases from the initial values. Neither the 4-agent nor 8-agent models perform as well as the Reputation model in terms of classification accuracy. Furthermore, consistent improvement as time continues does not occur with the stochastic model. Instead the classification accuracies remain sporadic and hover around the 80% and 83% marks for the 4-agent and 8-agent models respectively. The simulations also indicate a broad variance between classification accuracies, with classifications as good as 92% and as low as 70% after the initial results.

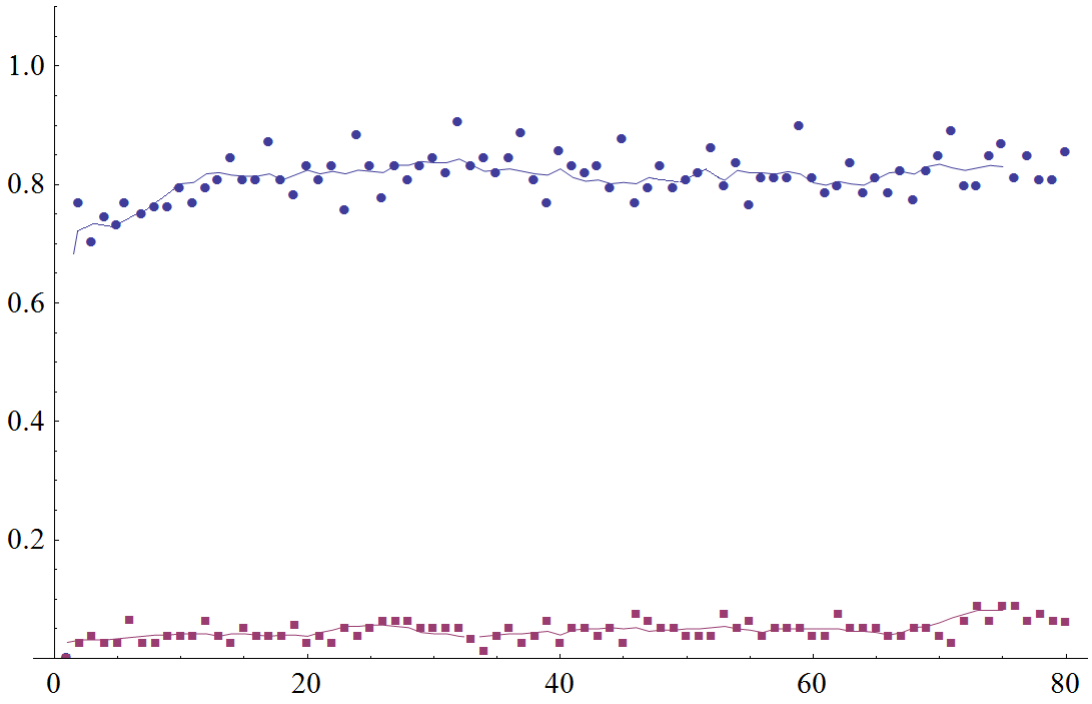

Figure 4.7: 4 Agent Stochastic Model: Accuracy & False Positives vs. time

We now evaluate the relative performance of Stochastic search model when using 4 or 8 agents in the network. Accuracy data for all 30 sample observations is provided in Table

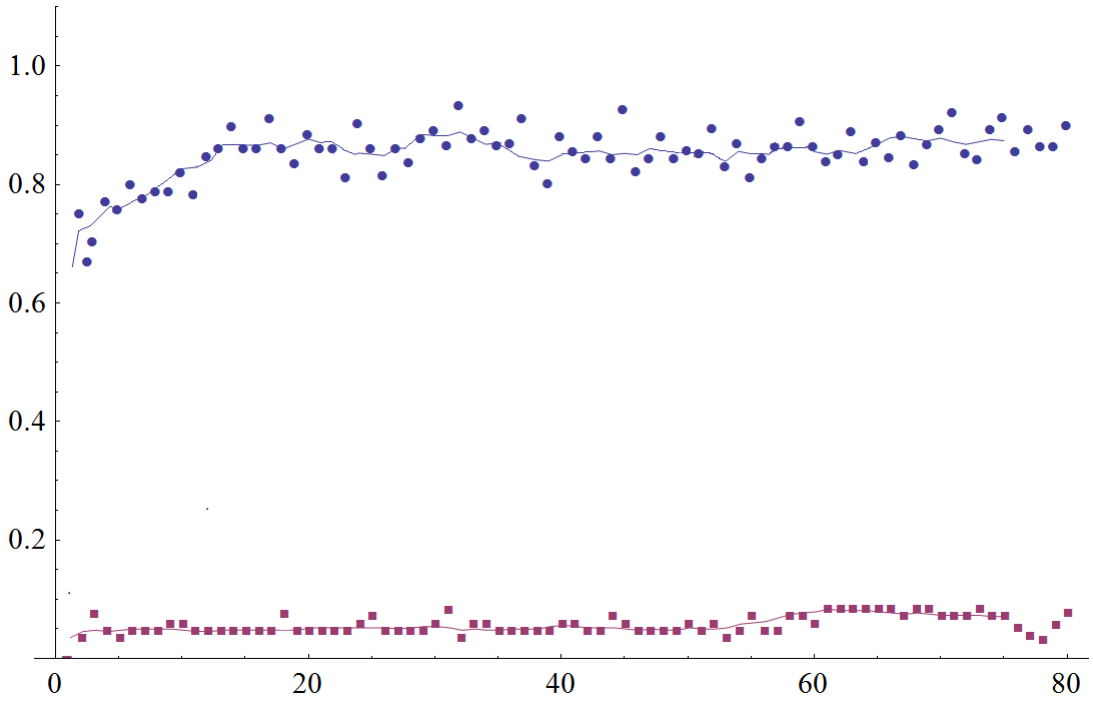

Figure 4.8: 8 Agent Stochastic Model: Accuracy & False Positives vs. time

4.6. The columns show results for each of the two experiments: 4-agent and 8-agent, and provides both the initial and final accuracy.

Table 4.6: Stochastic Search Overall Accuracy

	Initial: 4-agent	**Initial: 8-agent**	**Final: 4-agent**	**Final: 8-agent**
mean	0.651	0.708	0.800	0.859
median	0.639	0.711	0.803	0.861
stddev	0.071	0.077	0.062	0.048

The Reputation model outperforms the Stochastic search in classification accuracy. The the Stochastic movement is capable of further optimization that may improve locating

better nodes for classifying attacks. Furthermore, the Stochastic model places all decisions locally on the agents. Agents act completely independent of one another in terms of voting and movement, yet self-organize to avoid coexisting on the same node. The greatest benefit of the model is that agents can be removed or added to the system without affecting the abilities of the other agents to function.

4.6.3 MFIREv3 Deterministic Algorithm Performance Assessment.

The Deterministic Search with Maximum Cover is a new movement model designed to test the effectiveness of distributing knowledge amongst nodes. Testing the accuracy of the system in accordance with section 3.4.3 us to compare the effectiveness of the deterministic model with the other movement models.

Figure 4.9 shows the number of agents that move in each time period. Agent movement is locally decided based on greedily achieving the greatest "gain" determined by an agents location with respect to nodes covered, distance from perceived threat, and amount of time the node remained unevaluated. The movement of agents stabilizes at around two almost immediately. This is due to the agent's desire to visualize the maximum number of nodes possible. Agents only seek new locations when the gain of seeking an un-searched node surpasses the gain of the maximum cover, or an agent detects an attack.

The number of false positive classifications increased dramatically for the deterministic model. The false positive classification averages 14% and 18% for the 4-agent and 8-agent models respectively, since only one agent is required to classify the network as under attack. Although the false positive rate is relatively high, the accuracy of the attacks classified remained good throughout the experiment. Table 4.7 provides the overall classification accuracies for both models.

The accuracy of the classifier is misleading, as far more normal scenarios are classified as attacks. However, when an attack takes place it is quickly classified as only one agent needs to recognize the attack.

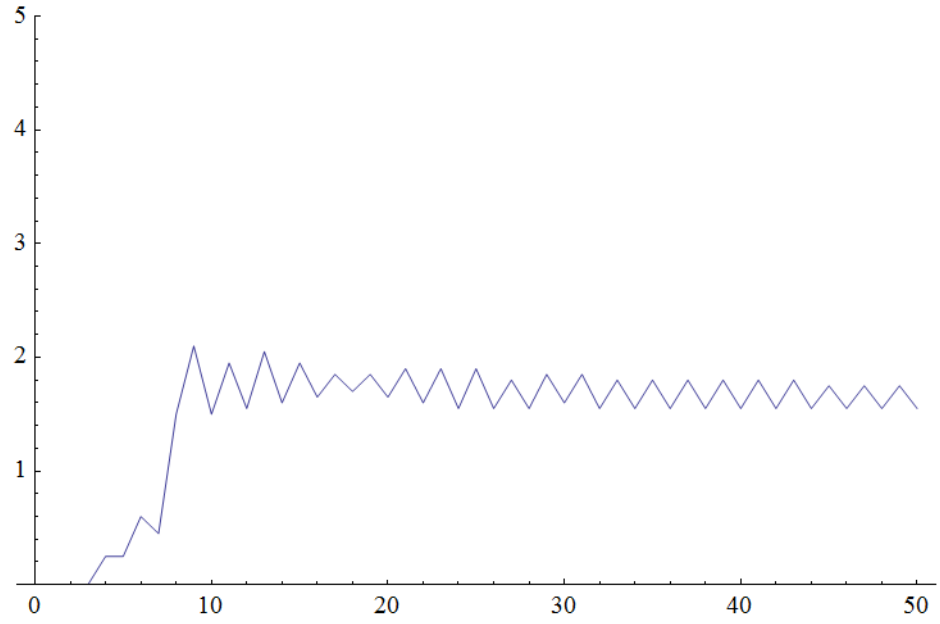

Figure 4.9: Average Agents Moving (8-agent Deterministic Model)

Table 4.7: Deterministic Search Overall Accuracy

	Initial: 4-agent	**Final: 4-agent**	**Initial: 8-agent**	**Final: 8-agent**
mean	0.744	0.821	0.889	0.912
median	0.751	0.833	0.878	0.922
stddev	0.032	0.036	0.041	0.049

The classification accuracies are good when the network is under attack, since only one agent needs to recognize the attack. Furthermore, the agents tend to monitor the network nodes with the most connections to visualize the entirety of the network. These results provide an alternative method for classifying network intrusions that could augment common Intrusion Detection approaches.

The movement of the agents within the network creates large data problems. With each node requiring an update each timestep as to what value it is since its last observation, validating the agent classifications takes a large amount of time. The calculations on the 100-node simulated network severely inhibited the performance of the agents with respect to time required.

The agents stay near the most actively connected nodes within the network each timestep, and as previously stated achieve good classification results. Making the agents motionless and placing them directly on the most connected nodes would create the same effect without creating an artifact of information on nodes visited.

The lack of an effective voting scheme lead to the high false-positive rate, however this voting system is required when the agent objective is to spread away from other agents in order to visualize the greatest number of uniques nodes. Testing effectively illustrated that quality classifications occur when agents maintain a broad view of the network, however node knowledge requires more computational complexity than desired for an effective MAS. The use of stationary agents located at highly connected nodes provides more research opportunities than the use of distributed knowledge at nodes. Stationary and mobile agents in cooperation with one another could utilize a variation of the Reputation model's voting system to improve network classification.

This concludes the testing and validation of the three MFIREv3 classification models.

4.7 Defense Analysis

The automated defenses of the agents in the MFIREv3 system is not meant to stop all attacks or even all instances of one attack. Attacking and defending networks is a probabilistic balance. As discussed in sections 2.9 and 3.5; the aim of network defense is to decrease the probabilities that: an attacker desires attacking the network, and that an attack on the network is successful. No defensive system is immune to every attacks, however bolstering the defenses increases the probability that the network remains safe.

Rate limiting greatly reduced the effective spread of the DDoS attack in the MFIREv3 system. Qualitatively validating the effectiveness of the rate limiting defensive measure illustrates that it is effective in preventing the DDoS attack from overflowing its target node.

In the 30 observations taken of the DDoS attack with rate limiting on the network, each consisting of 20 simulations, the attack failed to reach the target node 97% of the time. In the instance where the attack was successful in reaching the target node, the attackers were located within a very short distance of the target node, making flooding the target significantly easier. This result qualitatively validates that rate-limiting is an effective tool in defense against DDoS attacks. Figure 4.10 illustrates how rate limiting greatly reduces the spread of the DDoS attack within the MFIREv3 framework.

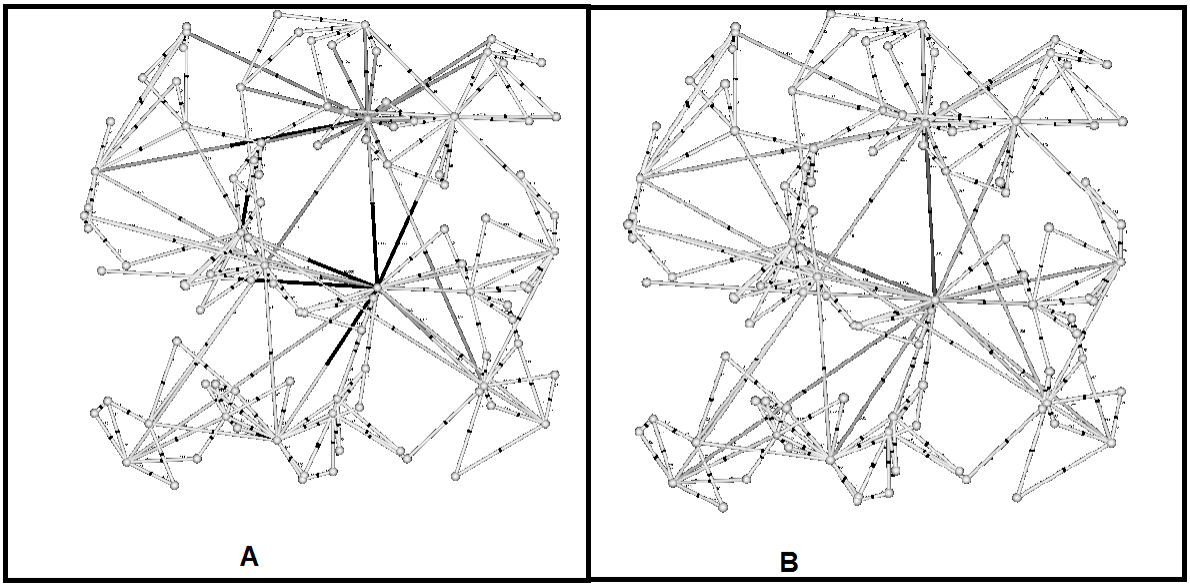

Figure 4.10: Figures A: Illustrating the effective spread of a DDoS attack without Rate Limiting and B: Illustrating Rate Limiting's effectiveness against DDoS attacks

The elimination of ports on network nodes in the presence of a Worm attack greatly improves the ability to protect vital resources in the network [127]. The objective of Worm containment defense is to qualitatively validate the ability to contain Worm Propagations by blocking all network traffic through nodes connecting infected areas to the rest of the network.

Section 3.5 illustrated the simple network used to validate the defensive system. At a minimum the Worm attack compromises three nodes due to the structure of the network's interconnected areas. If the attack expands beyond the initial quadrant it attacked, the defensive system fails the observation.

With 30 observations consisting of 20 simulations, we test the elimination of node ports by attacking one of four locations with a Worm propagation. The agents prevented the spread of the Worm propagation in 28 of the 30 observations. The success of the system is qualitatively validated on the test network. The defensive system can be implemented on more complex networks with larger numbers of nodes to determine its utility in structures without clear areas to contain an attack.

4.8 Summary

This chapter details the experimental results of our research effort. The chapter begins with a discussion of proper software testing and validation of results. Feature selection testing and analysis follows in Section 4.3. All three movement model experimental designs and analysis occur before the discussion of Defensive methods design and analysis concludes the chapter.

V. MFIREv3 Conclusions and Future Research

This chapter highlights the successes and opportunities resulting from MFIREv3 concerning the continuing development of a multi agent system for flow-based intrusion detection and cyber security. Section 5.1 discusses the observations of meeting the objectives proposed in Section 1.2. Section 5.2 highlights opportunities for future research activity. Potential Real World applications of the MFIREv3 system are presented in 5.3. Finally an overall research summary is presented.

5.1 Conclusions

The critical need for research investigations into distributed, flow-based intrusion detection systems is stated in Chapter 1. As noted by Sperotto et al. [163], "distributed detection is particularly important, because the amount of traffic on high-speed networks is still increasing, suggesting that scalability remains an issue in the future." Furthermore, the ability to defend these attacks is of increasing importance, as the risk-reward scenario favors attacking valuable assets at this point in time.

The *goal* of this research is to develop a scalable distributed Multi Agent System (MAS) for the defense of flow based system attacks and anomaly detection and identification. The following high-level *objectives* from Chapter 1.2 support this cyber security goal:

1. Continue design and evaluate a multi-agent intrusion detection system using a Reputation system

2. Evaluate the MFIREv2 multi-agent intrusion detection system using stochastic search

3. Design and evaluate a multi-agent intrusion detection system using deterministic search with search incentives and Maximum Cover

4. Design and evaluate a Multi Objective Evolutionary Algorithm for best subset feature selection

5. Determine if attacks can be classified using a Linear Kernel as opposed to the Radial Basis Function when using MOEA selected features

6. Create a robust distributed simulation framework for evolving self organizing multi-agent systems

7. Create a robust simulation framework for the automated defense of the network

We are successful in achieving all of the listed objectives. Our successful research effort develops a unique multi agent system designed to engage in flow-based intrusion detection in a distributed fashion. The achievement of the listed measurable objectives illustrates completion of our research *goal*.

The fully distributed system eliminates the single point of failure that existed with the centralized controller. This functionality was validated through the elimination of a single agent during a test, but extensive experimentation on the accuracy of the system was not conducted. Testing of systems with changing numbers of agents was conducted by Eric Holloway with SOMAS [84].

Objective 1 is achieved by the experimental design described in Section 4.5.1 and the analysis of the results in Section 4.6.1. The results illustrate not only the completion of the objective, but also that of the current MFIREv3 Reputation model achieves a 94% average classification accuracy using 8-agents.

Objective 2 is met by the experimental design described in Section 4.5.2 and the analysis of the results in Section 4.6.2. The completion of the evaluation of a Stochastic Search technique shows that the use of genetic algorithms to influence movement actuators provides a viable solution to a fully distributed MAS. Although the model is not optimized for comparable results to the Reputation model at the current time, it presents a faster

approach that with further research could rival the Reputation model in finding good agent locations for attack classification.

Objective 3 is achieved by the experimental design described in Section 4.5.3 and the analysis of the results in Section 4.6.3. The deterministic search model provides an additional avenue of research. The model provides good vantage points for a large portion of the network when the agents approach Maximum Cover nodes. In the large and complex environment of a network, *node reputation* presents big data issues. The use of a similar model in a larger network requires network specific agent locations over smaller areas. Another option is the use of stationary agents located at predetermined Maximum Cover locations working in conjunction with moving agents. Stationary agents achieve the *birds-eye view* of the network missing from the movement models that aim to move all agents towards attacks, without the big data.

The increase in classification accuracy over the previous effort, MFIREv2 [185], is directly attributable to the use of feature selection. In both the baseline and Reputation model, classification accuracies surpassed the previous effort. Section 4.3 indicates that the MOEA for feature selection designed in Section 3.3.3 provided a good set of features to improve the classification accuracy of the system. These results indicate the completion of Objective 4.

Objective 5 aims to improve the efficiency of classification using a Linear Kernel to classify network attacks. Using the features selected by the MOEA, Section 4.4 tested the accuracy in classifying individual attacks. The results supported the use of a Radial Basis Function, as the data is not linearly separable until it is mapped to a higher dimensional space. Appendix F contains the rest of the graphical results of all of the Linear Kernel Tests mot included in Section 4.4. The testing provided deeper knowledge into the clustering of our datasets.

Self-organization is the ability of a system to adapt in pursuit of better performance. The multi agent system reconfigures itself (in terms of spatial distribution of the agents) through the use of reputation, stochastic free-movement, and Multi-Objective search. This testing, discussed in Section 4.1, validated the completion of Objective 6. This is because the performance gains in certain conditions are attributable to the use of these models, the multi agent systems exhibit evidence of self-organization.

Section 4.7 illustrates the testing and analysis of network defenses against simulated attacks. The results of the defense methods of the agents indicate that agents are capable of containing and slowing attacks autonomously in a flow based simulation. The ability for a mobile agent to recognize an attack and attempt to prevent the spread of and shutdown the attack is important. The ability to contain an attack protects network resources and has the potential to allow the user to detect the origin of the attack. The possibility of potential criminal prosecution provides an additional psychological deterrence for the system defenses. The results of Section 4.7 indicate the completion of our final objective.

With the completion of these objectives, we achieve the *goal* of developing a distributed MAS for the defense of flow based system attacks and anomaly detection and identification.

5.2 Future Research Activity

The most immediate need for future research is a total, comprehensive performance evaluation of MFIREv3. Factorial design can investigate the effects of various factors individually and jointly for increased understanding of how to set system parameters for effective performance. The objective is to determine the effectiveness of the support vector machine for classification in this domain.

One of the products that evolved out of our research is a set of network simulation model refinements and extensions that, if pursued, may increase the range and depth of MFIREv3 impact. These include:

- Agents with adaptability to set different feature subsets for classifying attacks based on previous network observation
- The voting system could be changed to a probabilistic model as opposed to the current one-vote/best-guess system.
- Adaptive numbers of agents as illustrated in SOMAS [84].

Finally, a very good area of future research is the use of both mobile and stationary agents on the same network. The Max Cover Algorithm illustrated that a single agent can make a good classification solely from a network hub. The distribution of knowledge is more effective if certain agents monitor the major traffic areas while others seek to optimally classify attacks at specific locations. The reputation/voting model could be completely modified to include combinations of what a local agent sees with respect to what its closest stationary agents sees. This model would effectively use a variation of the Reputation model's voting system, with stationary agents as well as mobile agents.

MFIREv3 developed from an object oriented framework, making these improvements readily incorporable. It can be expanded or focused as needed to run countless conceivable experiments with Multi-Agent systems, Intrusion Detection, and network or attack simulation.

5.3 Applications in Real-Word Settings

MFIREv3 presents supplementary methods to bolster any network's cyber security systems. Signature-based intrusion detection system libraries allow for rapid classification of known network attacks, however when these systems fail an anomaly-based detection system may still be capable of detecting the attack.

Failures arise with signature-based IDSs when new attack implementations exploit a previously unknown vulnerability, known as a *zero-day attack* [129]. Attacks such as StuxNet and Flame bypassed common IDSs with ease [7]. Although the MFIREv3 system

is not deployed on active networks, and StuxNet and Flame present advanced capabilities that do not compare with standard attacks, it is possible that StuxNet and Flame create anomalous flows detectable by a system such as MFIREv3.

MFIREv3's application to real-world intrusion detection is that it does not require an updated library of known attack signatures. If a *zero-day* worm creates a flow that the system can recognize, then the anomaly-based IDS detects it and thwarts an otherwise successful intrusion.

5.4 Overall Summary

This chapter began with a review of the research goals and objectives and their completion. Following, we provided a path for research into future iterations of MFIREv3 and multi agent, flow-based intrusion detection systems.

This research effort showed that given the objectives listed in Section 1.2, the effectiveness of the intrusion detection and classification system is improved. This is an important contribution that supports our goal of developing an effective, flow-based, multi agent system for inter-AS network attack classification.

The field of cyber security research for distributed, flow-based intrusion detection continues to be wide open for exploration. In the fight to protect the inestimable advantages of our networks of autonomous systems, whether for civilian applications or in support of operations conducted by the United States military with which we are affiliated, one cannot afford to overlook multi-agent, flow-based intrusion detection techniques. It is our hope that MFIREv3 can serve as an inspiration for parallel efforts, with the possibility of creating an effective and scalable complement to a suite of cyber network defense capabilities.

Appendix A: History

This appendix details the development of the MFIRE Simulator from its beginnings to its current version: MFIREv3.0

A.1 SOMAS

Self Organized Multi Agent Swarms (SOMAS) was created by Eric Holloway to study the effects of a dynamic, decentralized intrusion defense system [84]. The multi agent system is formally modeled as a DEC-POMDP, a I-POMDP, and a new F(*-POMDP). Agents in the network are evolved using a multi-objective genetic algorithm. These agents have the ability to change location, instantiate other agents and delete agents, as well as various methods to modify GA chromosomes and fitness values. Also, enemy agents have additional methods of stealing or corrupting data on a node, sending denial-of-service packets, compromising a node, and others. These functions are activated by actuators, which get their input from rules and sensors. The relationship between the sensors, rules and actuators are optimized by the GA, which allows agents to defend against threats in the network. The agents learn to defend against attacks in a number of pre-defined scenarios, including: Intrusion Elimination, Enemy Avoidance, DDoS, and Information War. The primary goal of SOMAS is to evaluate the effectiveness of self-organization and "entangled hierarchies" for accomplishing scenario objectives. One of the interesting features of SOMAS is the ability for agents to take active defensive action in the network, rather than simply passively detecting an attack. For a complete description of SOMAS, see [84].

A.2 MFIRE v1.0

MFIRE 1.0 was created by Capt David Hancock as a network simulation environment to conduct flow-based intrusion detection experiments using a reputation-based multiagent

system [75–77]. One critique of SOMAS was its rudimentary implementation of network topology and routing. MFIRE 1.0 was an attempt to create a more realistic simulation environment. MFIRE 1.0 is written in Java, and makes use of the MASON DES, and TopGen network topology generator. Networks and gateway routers are simulated down to the Autonomous System level, and packet routing is a faithful implementation of the Border Gateway Protocol. Delays and packet loss are handled by the system to a reasonable level of realism, while still allowing the DES to simulate a large network at a fast rate. In addition to the network components (nodes, links and packets), the prominent high-level objects are processes, observations and classifiers. A process object allows arbitrary code to run on any node. Subclasses derived from processes are made into agents, attackers, background internet traffic, etc. In addition, each node collects flow-based observations based on the current and past network traffic. Agents create features from these observations, which are used to classify if an attack is occurring. Agents may use any user-defined classifier. Finally a Reputation system for the MAS allows the providers to rate the reliability of each agent's classifications. This system prompts the agents to move to better vantage points within the network, and imparts self-organization to the MAS. Special attention is paid to the design of the MAS communication, which allows inter-agent communication in the presence of many types of faults (see Appendix A). The provided hierarchical class structure allows the framework to be extended for many different experiment types, using an object-oriented approach.

Capt Hancock [77] presents the hypothesis that a flow-based, multi-agent network attack classifier can be made more effective by:

1. employing a reputation system to govern agent mobility
2. adding a decay factor to each agent's reputation to further spur agents to find nodes providing the most "useful" information

From this hypothesis, four objectives are defined:

1. *Develop an effective network simulation environment appropriate for the problem scope.*

2. *Validate the proper functioning of simulated malicious traffic.*

3. *Validate the proper command, control, and communications in the multi agent intrusion detection system.*

4. *Study the effects of several factors on classification accuracy.*

The first three objectives are qualitatively validated with MFIRE 1.0.

A.3 MFIRE v2.0

Captain Timothy Wilson continued the research hypothesis of Hancock in MFIREv2 [185]. The main focus was to study the effects of several factors on classification accuracy. Wilson conducted this testing by applying different numbers of agents to the system.

Wilson introduced a *free-movement model*, that eliminated the need of a central reputation controller. This system did not undergo accuracy analysis, but provided a new system for classification testing. For a complete description of MFIREv2, see [185].

Appendix B: Network Threats

Within these categories, many types of intrusion are recognized and five of them are evaluated in this research [126]:

Information Gathering—Network devices can be discovered and profiled in much the same way as other types of systems. Attackers usually start with port scanning. After they identify open ports, they use banner grabbing and enumeration to detect device types and to determine operating system and application versions. Armed with this information, an attacker can attack known vulnerabilities that may not be updated with security patches.

Sniffing—Sniffing or eavesdropping is the act of monitoring traffic on the network for data such as plaintext passwords or configuration information. With a simple packet sniffer, an attacker can easily read all plaintext traffic. Also, attackers can crack packets encrypted by lightweight hashing algorithms.

Spoofing—Spoofing is a means to hide one's true identity on the network. To create a spoofed identity, an attacker uses a fake source address that does not represent the actual address of the packet. Spoofing may be used to hide the original source of an attack or to work around network access control lists (ACLs) that are in place to limit host access based on source address rules.

Session Hijacking—Also known as man-in-the-middle (MitM) attacks, session hijacking deceives a server or a client into accepting the upstream host as the actual legitimate host. Instead the upstream host is an attacker's host that is manipulating the network so the attacker's host appears to be the desired destination.

Denial of Service—Denial of service denies legitimate users access to a server or services. The SYN flood attack is a common example of a network level denial of service attack. It is easy to launch and difficult to track. The aim of the attack is to send more requests to a server than it can handle. The attack exploits a potential vulnerability in the

TCP/IP connection establishment mechanism and floods the server's pending connection queue.

Viruses, Trojan Horses, and Worms—A virus is a program that is designed to perform malicious acts and cause disruption to the operating system or applications. A Trojan horse resembles a virus except that the malicious code is contained inside what appears to be a harmless data file or executable program. A worm is similar to a Trojan horse except that it self-replicates from one server to another. Worms are difficult to detect because they do not regularly create files that can be seen. They are often noticed only when they begin to consume system resources because the system slows down or the execution of other programs halt. The Code Red Worm is one of the most notorious to afflict IIS; it relied upon a buffer overflow vulnerability in a particular ISAPI filter. The success of these attacks on any system is possible through many vulnerabilities such as weak defaults, software bugs, user error, and inherent vulnerabilities in Internet protocols.

Footprinting—Examples of footprinting are port scans, ping sweeps, and NetBIOS enumeration that can be used by attackers to glean valuable system-level information to help prepare for more significant attacks. The type of information potentially revealed by footprinting includes account details, operating system and other software versions, server names, and database schema details.

Password Cracking—If the attacker cannot establish an anonymous connection with the server, he or she will try to establish an authenticated connection. For this, the attacker must know a valid username and password combination. Unchanged default account names, and the use of blank or weak passwords makes the attacker's job even easier.

Arbitrary Code Execution—If an attacker can execute malicious code on the server, the attacker can either compromise server resources or mount further attacks against downstream systems. The risks posed by arbitrary code execution increase if the server process under which the attacker's code runs is over-privileged. Common vulnerabilities

include weak IIS configuration and unpatched servers that allow path traversal and buffer overflow attacks, both of which can lead to arbitrary code execution.

Unauthorized Access—Inadequate access controls could allow an unauthorized user to access restricted information or perform restricted operations. Common vulnerabilities include weak IIS Web access controls, including Web permissions and weak NTFS permissions.

Hansman and Hunt [78] provide a short taxonomy of network and computer attacks listed in Figure B.1.

Table 1 The first dimension's categories

Level 1	Level 2	Level 3
Viruses:	File infectors	
	System/boot record infectors	
	Macro	
Worms:	Mass mailing	
	Network aware	
Buffer overflows:	Stack	
	Heap	
Denial of service attacks:	Host-based:	Resource hogs
		Crashers
	Network-based:	TCP flooding
		UDP flooding
		ICMP flooding
	Distributed	
Network attacks:	Spoofing	
	Session hijacking	
	Wireless attacks:	WEP cracking
	Web application attacks	Cross site scripting
		Parameter tampering
		Cookie poisoning
		Database attacks
		Hidden field manipulation
Physical attacks:	Basic	
	Energy weapon:	HERF
		LERF
		EMP
	Van Eck	
Password attacks:	Guessing:	Brute force
		Dictionary attack
	Exploiting implementation	
Information gathering attacks:	Sniffing:	Packet sniffing
	Mapping	
	Security scanning	

Figure B.1: Short Taxonomy of Attacks [78]

Once again, this is not a plethoric list of possible attacks, but does list the common categories of attacks on networks. The attacks examined in this research effort include: Denial of Service, Worm, Scan, Trojan, and Man-in-the Middle. These represent a list

of attacks that create flow-based anomalies in the network in both a local and distributed fashion.

For an exhaustive taxonomy of network threats, see Lough's dissertation on a taxonomy of computer attacks [114]

Appendix C: Popular DES Engines

Some of the more well-known DES options and their areas of emphasis are:

- OMNeT++, [174]: network simulation
- MASON, [115]: agent-based systems simulation
- CNET, [122, 123]: network simulation
- GloMoSim, [191]: large-scale wireless networks
- OPNET : network simulation
- NS2, [121]: network simulation
- PARSEC, [12]: parallelization
- SystemC : electronics systems-level modeling
- Tortuga : general DES with Java/Eclipse integration
- SimPy: general DES for Python

Appendix D: Popular SVM Packages

A subset of popular SVM packages [89]:

- SVM^{light} [91]: SVM^{light}, by Joachims, is one of the most widely used SVM classification and regression packages. It has a fast optimization algorithm, can be applied to very large datasets, and has a very efficient implementation of the leave-one-out cross-validation. Distributed as C++ source and binaries for Linux, Windows, Cygwin, and Solaris. Kernels: polynomial, radial basis function, and neural (tanh).

- LibSVM [36]: LibSVM (Library for Support Vector Machines), is developed by Chang and Lin and contains C-classification, v-classification, and ϵ-regression. Developed in C++ and Java, it supports also multi-class classification, weighted SVM for unbalanced data, cross-validation and automatic model selection. It has interfaces for Java, Python, R, Splus, MATLAB, Perl, Ruby, and LabVIEW. Kernels: linear, polynomial, radial basis function, and neural (tanh).

- SVMTorch: SVMTorch, by Collobert and Bengio, is part of the Torch machine learning library and implements SVM classification and regression. Distributed as C++ source code or binaries for Linux and Solaris.

- Weka: Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from a Java code. Contains an SVM implementation.

- SVM in R: This SVM implementation in R (http://www.r-project.org/) contains C-classification, n-classification, e-regression, and n-regression. Kernels: linear, polynomial, radial basis, neural (tanh).

- MATLAB SVM Toolbox: This SVM MATLAB toolbox, by Gunn, implements SVM classification and regression with various kernels: linear, polynomial, Gaussian radial basis function, exponential radial basis function, neural (tanh), Fourier series, spline, and B spline.

- TinySVM: TinySVM is a C++ implementation of C-classification and C-regression which uses sparse vector representation and can handle several ten-thousands of training examples, and hundred-thousands of feature dimensions. Distributed as binary/source for Linux and binary for Windows.

- Spider: Spider is an object orientated environment for machine learning in MATLAB, for unsupervised, supervised or semi-supervised machine learning problems, and includes training, testing, model selection, cross-validation, and statistical tests. Implements SVM multi-class classification and regression.

- jlibsvm [160]: Heavily refactored Java port of LibSVM. Implements optimized kernel functions using Java class structure and APIs, and has support for multithreaded training.

Appendix E: Intrusion Detection System Details

Concepts of classifying intrusions require terminology and metrics. For ease of readership this list provides many of the terms used in quantitatively evaluating the quality of the MFIRE system.

Some terminology and important concepts for IDSs are as follows [182]:

- *Alert/Alarm*: A signal suggesting that a system has been or is being attacked.
- *True Positive*: A legitimate attack which triggers an IDS to produce an alarm.
- *False Positive*: An event signaling an IDS to produce an alarm when no attack has taken place.
- *False Negative*: A failure of an IDS to detect an actual attack.
- *True Negative*: When no attack has taken place and no alarm is raised.
- *Noise*: Data or interference that can trigger a false positive.
- *Site policy*: Guidelines within an organization that control the rules and configurations of an IDS.
- *Site policy awareness*: An IDS's ability to dynamically change its rules and configurations in response to changing environmental activity.
- *Confidence value*: A value an organization places on an IDS based on past performance and analysis to help determine its ability to effectively identify an attack.
- *Alarm filtering*: The process of categorizing attack alerts produced from an IDS in order to distinguish false positives from actual attacks.

- *Attacker or Intruder*: An entity who tries to find a way to gain unauthorized access to information, inflict harm or engage in other malicious activities.
- *Masquerader*: A user who does not have the authority to a system, but tries to access the information as an authorized user. They are generally outside users.
- *Misfeasor*: They are commonly internals who misuse their powers
- *Clandestine user*: A user who acts as a supervisor and tries to use his privileges so as to avoid being captured.

Appendix F: Kernel Tests

This section provides the data of the five attack scenarios using the "optimal" features to create separability from normal attack traffic. The results visually illustrate the attacks non-linear separability. This test illustrates the need for a higher dimensional support vector machine kernel function. For this effort we utilize a Gaussian Kernel.

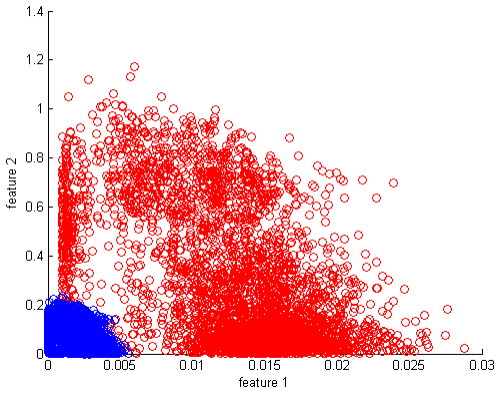

Figure F.1: Illustration of DDoS vs Normal Features F1: Num inbound Bytes F2: Num inbound Packets

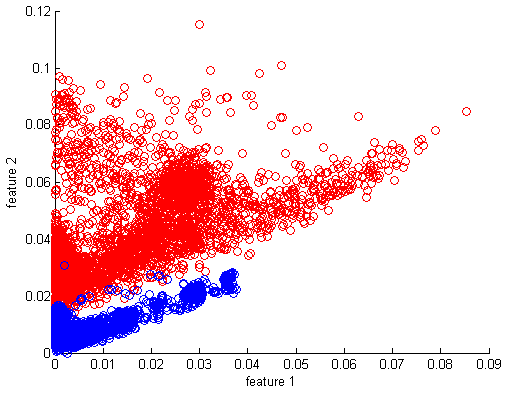

Figure F.2: Illustration of Scan vs Normal Features F1: Num distinct dest addrs F2: Num distinct dest ports

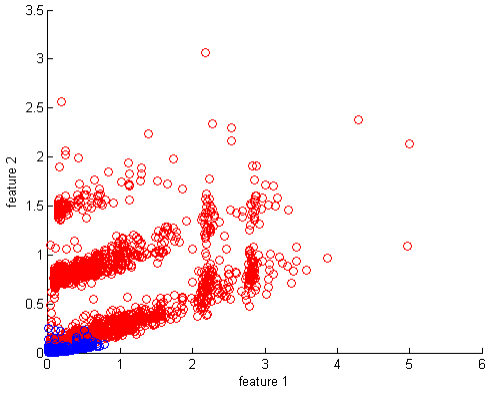

Figure F.3: Illustration of Worm vs Normal Features F1: Ratio of packets to dest tuples F2: Ratio of packets from source tuples

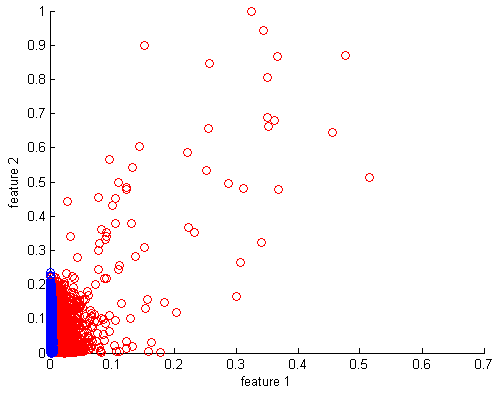

Figure F.4: Illustration of MitM vs Normal Features F1: Local Num inbound Bytes F2: Ratio of source ports to addr

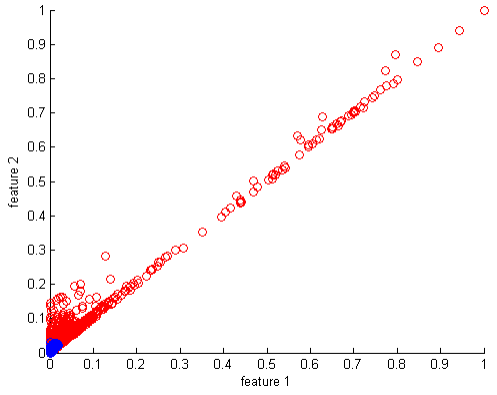

Figure F.5: Illustration of Trojan vs Normal Features F1: Num distinct source ports F2: Num inbound packets

Appendix G: MFIRE System Details

This appendix provides some additional MFIRE details. In section G.1, the messages used by MFIRE are detailed. Section G.2 lists the fourteen observations collected by MFIRE agents for use in derived features.

G.1 MFIRE: Messages

The figures provided in this section show the messages used in MFIRE. In each figure, the left side is used for the sender. The type of the message is displayed first, and below it, the format. The format is essential for extracting message components from the packet's payload, which itself is a single string. On the right side of each figure, we show the actions that are taken by the recipient.

Figure G.1 shows the messages sent from the controller and received by agents.

Figure G.2 shows the messages sent from agents and received by the controller.

Figure G.3 shows the SHARE message used for feature value exchange between agents.

Figure G.4 shows the messages involved in agent migration. MIGRATE is sent by an agent that received MOVE from the controller previously. It is sent to the AgentManager at the migration destination node. The MIGRATE message contains all information required to reinstantiate the agent at the distant end. MIGRATEACK is sent by an AgentManager that received a MIGRATE message previously. It is sent to the AgentManager at the node where the original copy of the migrating agent still resides. The AgentManager that receives MIGRATEACK terminates the agent.

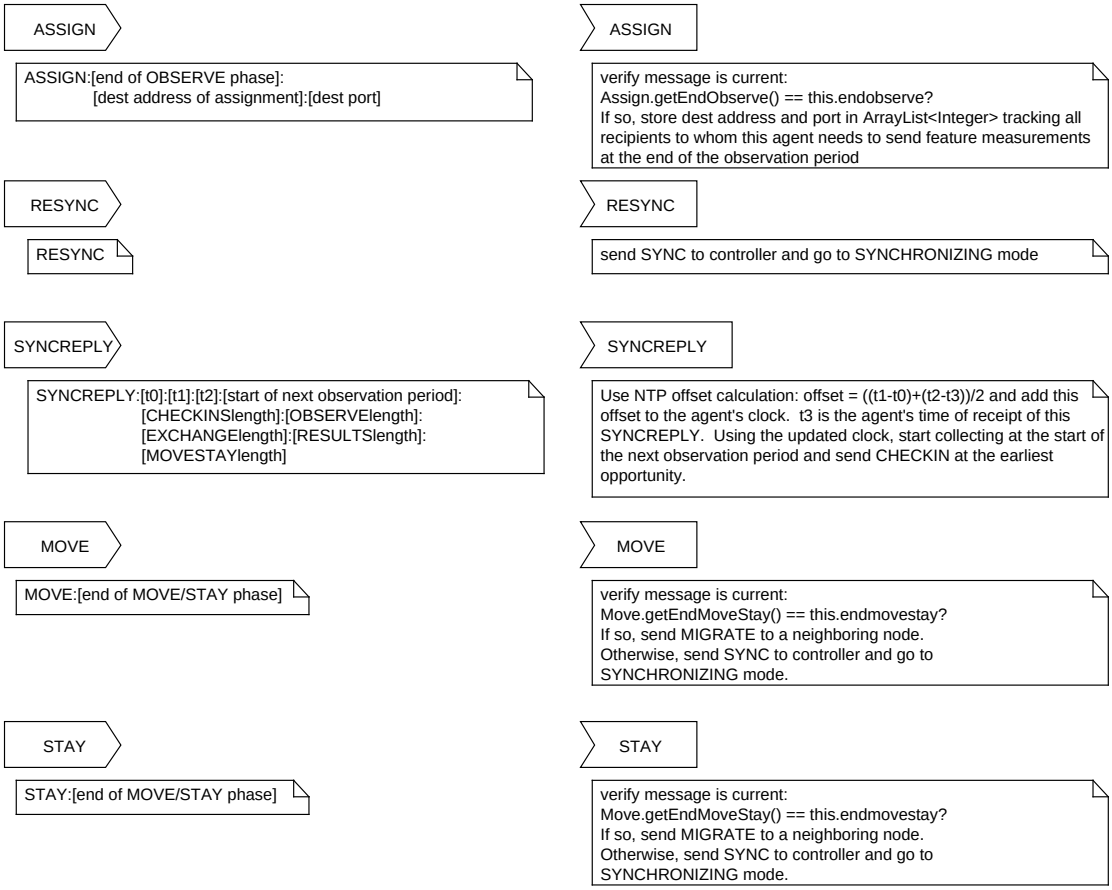

Figure G.1: MFIRE: Messages sent by the providers and received by agents

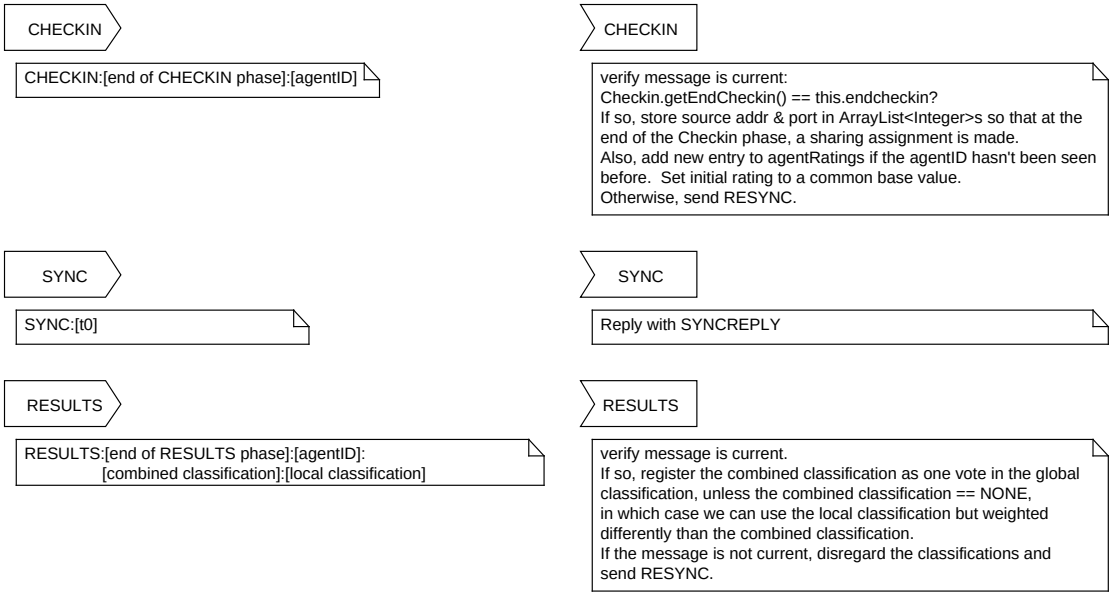

Figure G.2: MFIRE: Messages sent by agents and received by the providers

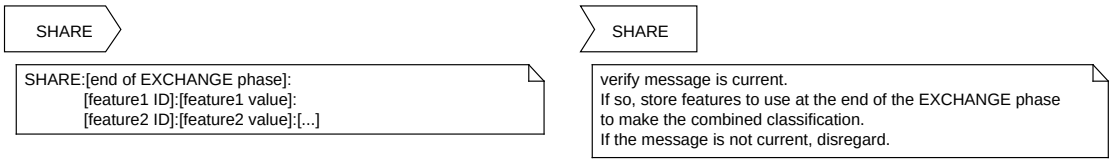

Figure G.3: MFIRE: Messages sent by agents to other agents

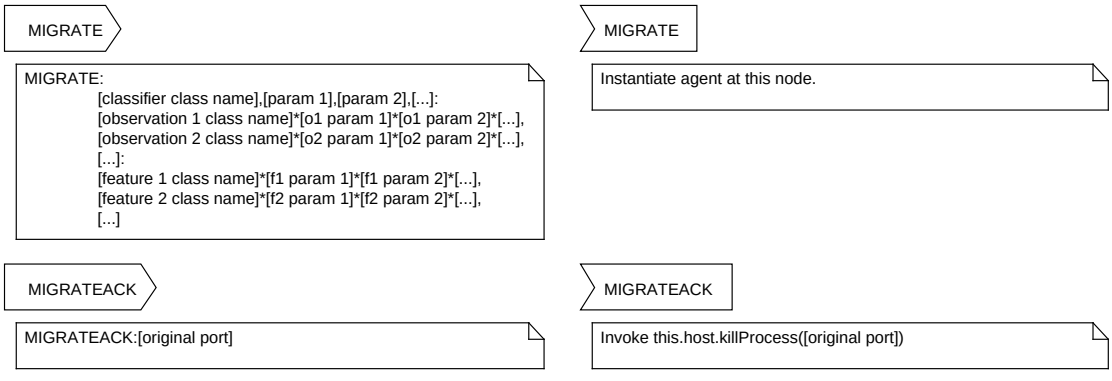

Figure G.4: MFIRE: Messages involved in agent migration

G.2 MFIRE: Observations

Each observation in MFIRE represents a traffic statistic collected over the duration of a single timestep. These are used to derive feature values.

The fourteen observations collected by agents in MFIRE:

1. Average number of bytes per $< destaddr, destport >$-tuple
2. Average number of bytes per $< sourceaddr, sourceport >$-tuple
3. Number of distinct destination addresses
4. Number of distinct $< destaddr, destport >$-tuples
5. Number of distinct destination ports
6. Ratio of destination ports to destination addresses
7. Total number of inbound bytes
8. Total number of inbound packets
9. Ratio of packets to $< destaddr, destport >$-tuples
10. Ratio of packets to $< sourceaddr, sourceport >$-tuples
11. Number of distinct source addresses
12. Number of distinct $< sourceaddr, sourceport >$-tuples
13. Number of distinct source ports
14. Ratio of source ports to source addresses

Clearly there are many linear dependencies in this set of observations. Care must be exercised when performing feature selection from this set.

Bibliography

[1] "Snort". URL www.snort.org.

[2] "Port Numbers", February 2011. URL http://www.iana.org/assignments/port-numbers.

[3] Achtert, C.; Kriegel H. P.; Kroger P.; Muller-Gorman I.; Zimek A., E.; Bohm. *Advances in databases concepts, systems and applications*. Springer, Berlin New York, 2007.

[4] Albert, R., H. Jeong, and A.L. Barabási. "Error and attack tolerance of complex networks". *Arxiv preprint cond-mat/0008064*, 2000.

[5] Amaldi, E. and V. Kann. "On the approximability of minimizing nonzero variables or unsatisfied relations in linear systems". *Theoretical Computer Science*, 209(1-2):237–260, 1998. ISSN 0304-3975.

[6] Anderson, Richard; Bozek Thomas, Robert H.; Brackney. "Advanced Network Defense Research", 2000.

[7] Asa, Norman. "Cyberattacks on Iran Stuxnet and Flame". *The New York Times*, 2013.

[8] Atserias, Albert, Andrei A. Bulatov, and Anuj Dawar. "Affine systems of equations and counting infinitary logic". *Theor. Comput. Sci.*, 410(18):1666–1683, 2009.

[9] Axelsson, S. "Intrusion detection systems: A survey and taxonomy". 2000.

[10] B, Neethu. "Classification of Intrusion Detection Dataset using machine learning Approaches". *International Journal of Electronics and Computer Science Engineering*, 1044–1065. Department of Computer Science, Amrita University, 2011. ISBN 2277-1956.

[11] Bäck, T. *Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms*. Oxford University Press, USA, 1996.

[12] Bagrodia, R., R. Meyer, M. Takai, Y. Chen, X. Zeng, J. Martin, and H.Y. Song. "Parsec: A parallel simulation environment for complex systems". *Computer*, 31(10):77–85, 2002. ISSN 0018-9162.

[13] Banks, J., B.L. Nelson, and D.M. Nicol. *Discrete-event system simulation*. Prentice Hall, 2009. ISBN 0136062121.

[14] Banzhaf, Peter; Keller Robert; Francone Frank, Wolfgang; Nordin. *Genetic programming : an introduction on the automatic evolution of computer programs and its applications*. Morgan Kaufmann Publishers Dpunkt-verlag, San Francisco, Calif. Heidelburg, 1998. ISBN 9781558605107.

[15] Bar, S., M. Gonen, and A. Wool. "A geographic directed preferential Internet topology model". *Computer Networks*, 51(14):4174–4188, 2007. ISSN 1389-1286.

[16] Barabási, A.L. and R. Albert. "Emergence of scaling in random networks". *Science*, 286(5439):509, 1999.

[17] Barford, P. and M. Crovella. "Generating representative web workloads for network and server performance evaluation". *Proceedings of the 1998 ACM SIGMETRICS joint international conference on Measurement and modeling of computer systems*, 160. ACM, 1998. ISBN 0897919823.

[18] Barr, R.S., B.L. Golden, J.P. Kelly, M.G.C. Resende, and W.R. Stewart. "Designing and reporting on computational experiments with heuristic methods". *Journal of Heuristics*, 1(1):9–32, 1995. ISSN 1381-1231.

[19] Bartos, Karel, Martin Grill, and Vojttech Krmicek. "Flow Based Network Intrusion Detection System using Hardware-Accelerated NetFlow Probes". *CESNET Conference 2008 Proceedings, pp. 49-56*. 2008.

[20] Becchi, M. "From Poisson Processes to Self-Similarity: a Survey of Network Traffic Models". 2008.

[21] Bellifemine, F., A. Poggi, and G. Rimassa. "JADE–A FIPA-compliant agent framework". *Proceedings of PAAM*, volume 99, 97–108. Citeseer, 1999.

[22] Bellifemine, F., A. Poggi, and G. Rimassa. "Developing multi-agent systems with a FIPA-compliant agent framework". *Software: Practice and Experience*, 31(2):103–128, 2001. ISSN 1097-024X.

[23] Bellman, R. "Adaptive control processes: a guided tour". *Princeton University Press*, 1:2, 1961.

[24] Berenger, Ralph D. "War in Cyberspace". *Journalism and Mass Communication*, 2010.

[25] Bertsekas, Dimitri. *Neuro-dynamic programming*. Athena Scientific, Belmont, Mass, 1996. ISBN 1886529108.

[26] Bhattacharyya, A. "On a measure of divergence between two statistical populations defined by probability distributions". *Bull. Calcutta Math. Soc*, 35:99–109, 1943.

[27] Bishop, C.M. *Pattern recognition and machine learning*, volume 4. Springer New York, 2006.

[28] Bollobás, B. *Random graphs*. Cambridge University Press, 2001.

[29] Brauckhoff, D., B. Tellenbach, A. Wagner, M. May, and A. Lakhina. "Impact of packet sampling on anomaly detection metrics". *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, 159–164. ACM, 2006. ISBN 1595935614.

[30] Brazier, Dunin-Keplicz B. Jennings N.R., F.M.T. and J. Treur. "DESIRE: Modelling Multi-Agent Systems in a Computational Formal Framework", 1997.

[31] Breitbart, Anthony. "Threat of next world war may be in cyberspace: UN". *Frost's Meditations*, 300(1):31–39, 2009.

[32] Bulatov, Andrei A. "H-Coloring dichotomy revisited". *Theor. Comput. Sci.*, 349(1):31–39, 2005.

[33] Burgess, Mark. "CFEngine". URL www.cfengine.com.

[34] Burnett, Chris, Timothy J. Norman, and Katia Sycara. "Sources of Sterotypical Trust in Multi-Agent Systems". *Trust in Agent Societies, 14th Edition*. 2011.

[35] Calvert, K.I., M.B. Doar, and E.W. Zegura. "Modeling Internet topology". *Communications Magazine, IEEE*, 35(6):160 –163, June 1997. ISSN 0163-6804.

[36] Chang, Chih-Chung and Chih-Jen Lin. *LIBSVM: a library for support vector machines*, 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm.

[37] Chapelle, O., P. Haffner, and V.N. Vapnik. "Support vector machines for histogram-based image classification". *Neural Networks, IEEE Transactions on*, 10(5):1055–1064, 2002. ISSN 1045-9227.

[38] Chella, Cossentino-M. Sabatucci L. Seidita V., A. "An agile process for designing agents", 2006.

[39] Chen, Z., L. Gao, and K. Kwiat. "Modeling the spread of active worms". *INFOCOM 2003. Twenty-Second Annual Joint Conference of the IEEE Computer and Communications. IEEE Societies*, volume 3, 1890–1900. IEEE, 2003. ISBN 0780377524. ISSN 0743-166X.

[40] Cheng, L. *Simulation and topology generation for large-scale distributed systems*. Ph.D. thesis, UNIVERSITY OF BRITISH COLUMBIA, 2009.

[41] Cheng, L., N.C. Hutchinson, and M.R. Ito. "RealNet: A topology generator based on real internet topology". *Advanced Information Networking and Applications-Workshops, 2008. AINAW 2008. 22nd International Conference on*, 526–532. IEEE, 2008.

[42] Cid, Daniel. "OSSEC". URL www.ossec.net.

[43] Coello, C.A.C., G.B. Lamont, and D.A. Van Veldhuizen. *Evolutionary algorithms for solving multi-objective problems*. Springer-Verlag New York Inc, 2nd edition, 2007.

[44] Cossentino, Fortino-G. Garro A. Mascillaro S. Russo W., M. "PASSIM: A Simulation-based Process for the Development of Multi-Agent Systems", 2008.

[45] Crucitti, Paolo, Vito Latora, Massimo Marchiori, and Andrea Rapisarda. "Error and attack tolerance of complex networks". *Physica A: Statistical Mechanics and its Applications*, 340(1-3):388 – 394, 2004. ISSN 0378-4371.

[46] Dawkins, Richard. *The Selfish Gene*. Oxford University Press, Oxford New York, 1989. ISBN 0192860925.

[47] De Wolf, T. "Analysing and Engineering Emergent Applications", 2007.

[48] Devroye, Luc. "Random variate generation in one line of code". *Proceedings of the 28th conference on Winter simulation*, WSC '96, 265–272. IEEE Computer Society, Washington, DC, USA, 1996. ISBN 0-7803-3383-7. URL http://dx.doi.org/10.1145/256562.256623.

[49] Doar, M.B. "A Better Model for Generating Test Networks". *Conference record*, 86. Institute of Electrical and Electronics Engineers, 1996.

[50] Duda, Richard O., Peter E. Hart, and David G. Stork. *Pattern Classification (2nd Edition)*. Wiley-Interscience, 2 edition, November 2000. ISBN 0471056693.

[51] Durillo, Juan J., Antonio J. Nebro, Francisco Luna, Bernabé Dorronsoro, and Enrique Alba. *jMetal: A Java Framework for Developing Multi-Objective Optimization Metaheuristics*. Technical Report ITI-2006-10, Departamento de Lenguajes y Ciencias de la Computación, University of Málaga, E.T.S.I. Informática, Campus de Teatinos, December 2006.

[52] Erdos, P. and A. Rényi. "On the evolution ofrandom graphs". *Publications of the Mathematical Institute of the Hungarian Academy of Science*, 5:17–61, 1960.

[53] Erriquez, Elisabetta. "An abstract framework for reasoning about trust". *Trust in Agent Societies, 14th Edition*. 2011.

[54] Fabrikant, A., E. Koutsoupias, and C. Papadimitriou. "Heuristically optimized trade-offs: A new paradigm for power laws in the Internet". *Automata, Languages and Programming*, 781–781, 2002.

[55] Faloutsos, M., P. Faloutsos, and C. Faloutsos. "On power-law relationships of the internet topology". *Proceedings of the conference on Applications, technologies, architectures, and protocols for computer communication*, 251–262. ACM, 1999. ISBN 1581131356.

[56] Feferman, Solomon. "The nature and significance of Godels incompleteness theorems", 2006.

[57] Ferber, J. *Multi-agent systems: an introduction to distributed artificial intelligence*. Addison-Wesley, 1999. ISBN 9780201360486. URL http://books.google.com/books?id=zt1SAAAAMAAJ.

[58] Ferreira, Candida. *Gene expression programming : mathematical modeling by an artificial intelligence*. Angra do Heroismo, Portugal, 2002. ISBN 9729589054.

[59] Fienberg, S. E. "Conflicts Between the Needs for Access to Statistical Information and Demands for Confidentiality". *Journal of Official Statistics*, 115–132. IEEE, 1994.

[60] Fisher, Danyel. "Animation for Visualization: Opportunities and Drawbacks", 2012.

[61] Floyd, R.W. "Algorithm 97: shortest path". *Communications of the ACM*, 5(6):345, 1962. ISSN 0001-0782.

[62] Franklin, S. and A. Graesser. "Is it an Agent, or just a Program?: A Taxonomy for Autonomous Agents". *Intelligent Agents III Agent Theories, Architectures, and Languages*, 21–35, 1997.

[63] Fudenberg, Drew. *Game theory*. MIT Press, Cambridge, Mass, 1991. ISBN 0262061414.

[64] Fujimoto, Richard M. "Parallel discrete event simulation". *Commun. ACM*, 33:30–53, October 1990. ISSN 0001-0782. URL http://doi.acm.org/10.1145/84537.84545.

[65] Gardelli, Viroli-M. Omicini A., L. "On the Roll of Simultions in the Engineering of Self-Organising MAS", July 2005.

[66] Garey, Michael. *Computers and intractability : a guide to the theory of NP-completeness*. W.H. Freeman, San Francisco, 1979. ISBN 0716710455.

[67] Gijsberts, Arjan. *Evolutionary Optimiztation of Kernel Machines*. Master's thesis, Delft University of Technology, August 2007.

[68] Gomez-Sanz, Botia J. Serrano E. Pavon J., J.J. "Testing and Debugging of MAS Interactions with INGENIAS", 2009.

[69] Gordon, J. "Pareto process as a model of self-similar packet traffic". *Global Telecommunications Conference, 1995. GLOBECOM'95., IEEE*, volume 3, 2232–2236. IEEE, 2002. ISBN 0780325095.

[70] Greene, William H. "Econometric Analysis", 1993.

[71] Grefenstette, John J. "Lamarckian Learning in Multi-agent Environments". *Proceedings of the Fourth International Conference on Genetic Algorithms*, 303–310. Morgan Kaufmann, 1991.

[72] Guyon, I. and A. Elisseeff. "An introduction to variable and feature selection". *The Journal of Machine Learning Research*, 3:1157–1182, 2003. ISSN 1532-4435.

[73] Haines, Joshua. "Extending the DARPA Off-Line Intrusion Detection Evaluations". *DISCEX-II*. Lincoln Laboratory, Massachusetts Institute of Techonology, 2011.

[74] Halbwachs, N. and L. Mandel. "Simulation and verification of asynchronous systems by means of a synchronous model". 2006. ISSN 1550-4808.

[75] Hancock, D. and G. Lamont. "Multi Agent Systems on Military Networks". *IEEE Symposium on Computational Intelligence in Cyber Security*. 2011.

[76] Hancock, D. and G. Lamont. "Reputation in a multi agent system for flow-based network attack classification". *IEEE Symposium on Intelligent Agents*. 2011.

[77] Hancock, David. *A Multi-Agent System for Flow-Based Intrusion Detection Using Reputation and Evolutionary Computation*. Master's thesis, Air Force Institute of Technology, March 2011.

[78] Hansman, Ray, Simon; Hunt. "A taxonomy of network and computer attacks". *Computer and Security*, 2004.

[79] Hastie, Trevor, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition (Springer Series in Statistics)*. Springer, 2nd ed. 2009. corr. 3rd printing edition, February 2009. ISBN 0387848576. URL http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

[80] Hernandez-Pereira, E., J.A. Suarez-Romero, O. Fontenla-Romero, and A. Alonso-Betanzos. "Conversion methods for symbolic features: A comparison applied to an intrusion detection problem". *Expert Systems with Applications*, 36(7):612–617, 2009.

[81] Herrero, Alvaro and Emilio Corchado. "Multiagent Systems for Network Intrusion Detection: A Review". *Computational Intelligence in Security for Information Systems*, 63:143–154, 2009.

[82] Hildebrandt, D., L. Bischofs, and W. Hasselbring. "RealPeer–A Framework for Simulation-Based Development of Peer-to-Peer Systems". *Parallel, Distributed and Network-Based Processing, 2007. PDP'07. 15th EUROMICRO International Conference on*, 490–497. IEEE, 2007. ISBN 0769527841. ISSN 1066-6192.

[83] Holdaway, Eric J. "ACTIVE COMPUTER NETWORK DEFENSE: AN ASSESSMENT", 2001.

[84] Holloway, Eric. *Self Organized Multi Agent Swarms for Network Security Control*. Master's thesis, Air Force Institute of Technology, March 2009.

[85] Horng, Shi-Jinn, M. Su, and Y. Chen. "A novel intrusion detection system based on hierarchical clustering and support vector machines". *Expert Systems with Applications*, 1:306–313, 2011.

[86] Howden, N., R. R
"onnquist, A. Hodgson, and A. Lucas. "JACK intelligent agents-summary of an agent infrastructure". *5th International Conference on Autonomous Agents*. Citeseer, 2001.

[87] Howell, Donna. "Tech Security 2013 Forecast: Clouds, Rain Of Threats". *Investor's Business Daily*, 2012.

[88] Huynh, T.D., N.R. Jennings, and N.R. Shadbolt. "An integrated trust and reputation model for open multi-agent systems". *Autonomous Agents and Multi-Agent Systems*, 13(2):119–154, 2006. ISSN 1387-2532.

[89] Ivanciuc, Ovidiu. "Applications of Support Vector Machines in Chemistry", 2007. URL http://www.support-vector-machines.org/SVM_soft.html.

[90] Jansen, W.A. "Intrusion detection with mobile agents". *Computer Communications*, 25(15):1392–1401, 2002. ISSN 0140-3664.

[91] Joachims, Thorsten. "SVM-light", August 2008. URL http://svmlight.joachims.org/.

[92] Josang, Audun, Roslan Ismail, and Colin Boyd. "A Survey of Trust and Reputation Systems for Online Service Provision". *Decision Support Systems*, 43:618–644, 2007.

[93] Karthick, R., Hattiwale V., and B. Ravindran. "Adaptive network intrusion detection system using a hybrid approach". *Fourth International Conference on Communication Systems and Networks (COMSNETS)*, 1:1–7, 2012.

[94] Kidney, Denzinger J., J. "Testing the Limits of Emergent Behavior in MAS using Learning of Cooperative Behavior".

[95] Kim, J., S. Radhakrishnan, and S.K. Dhall. "Measurement and analysis of worm propagation on Internet network topology". *Computer Communications and Networks, 2004. ICCCN 2004. Proceedings. 13th International Conference on*, 495–500. IEEE, 2005. ISBN 0780388143. ISSN 1095-2055.

[96] Kimeldorf, G., G.; Wahba. "Some Results on Tchebycheffian Spline Functions", 1971.

[97] Knuth, D.E. *The Stanford GraphBase: a platform for combinatorial computing*. AcM Press, 1993. ISBN 0201542757.

[98] Kohavi, R. and G.H. John. "Wrappers for feature subset selection". *Artificial intelligence*, 97(1-2):273–324, 1997. ISSN 0004-3702.

[99] Kong, J., M. Mirza, J. Shu, C. Yoedhana, M. Gerla, and S. Lu. "Random flow network modeling and simulations for DDoS attack mitigation". *Communications, 2003. ICC'03. IEEE International Conference on*, volume 1, 487–491. IEEE, 2003. ISBN 0780378024.

[100] Kotsiantis, S. B. "Supervised Machine Learning: A Review of Classification Techniques". *Informatica*, 31:249–268, 2007.

[101] Krishnamurthy, B. and W. Willinger. "What are our standards for validation of measurement-based networking research?" *ACM SIGMETRICS Performance Evaluation Review*, 36(2):64–69, 2008. ISSN 0163-5999.

[102] Kuipers, B.J. "Qualitative simulation: then and now". *Artificial intelligence in perspective, MIT Press, Cambridge, MA*, 1994.

[103] Kung, H. T., F. Luccio, and F. P. Preparata. "On Finding the Maxima of a Set of Vectors". *J. ACM*, 22(4):469–476, October 1975. ISSN 0004-5411. URL http://doi.acm.org/10.1145/321906.321910.

[104] Kurose, J. and K. Ross. *Computer Networking: A top-down approach*. Pearson Addison-Wesley, fifth edition, 2009.

[105] Lal, T., O. Chapelle, J. Weston, and A. Elisseeff. "Embedded methods". *Feature Extraction*, 137–165, 2006.

[106] Larman, C. *Applying UML and patterns: an introduction to object-oriented analysis and design and the unified process*. Prentice Hall PTR Upper Saddle River, NJ, USA, 2001. ISBN 0130925691.

[107] Lavelle, Stephen. "Fabricating Synthetic Data in Support of Training for Domestic Terrorist Activity Data Mining Research", 2010.

[108] Lawrence, Jeannette. *Introduction to neural networks : design, theory and applications*. California Scientific Software, Nevada City, Calif, 1994. ISBN 1883157005.

[109] Lee, Yi;, Yoonkyung; Lin and Grace Wahba. "Multicategory Support Vector Machines: Theory and Application to the Classi cation of Microarray Data and Satellite Radiance Data", 2004.

[110] Li, M. "An approach to reliably identifying signs of DDOS flood attacks based on LRD traffic pattern recognition". *Computers & Security*, 23(7):549–558, 2004.

[111] Liljenstam, M., Y. Yuan, BJ Premore, and D. Nicol. "A mixed abstraction level simulation model of large-scale Internet worm infestations". *Modeling, Analysis and Simulation of Computer and Telecommunications Systems, 2002. MASCOTS 2002. Proceedings. 10th IEEE International Symposium on*, 109–116. IEEE, 2003. ISBN 0769518400. ISSN 1526-7539.

[112] Lomuscio, A. and F. Raimondi. "MCMAS: A Model Checker for Multi-agent Systems", 2006.

[113] López, S., A. Brintrup, D. McFarlane, and D. Dwyer. "Selecting a multi-agent system development tool for industrial applications: a case study of self-serving aircraft assets". *Digital Ecosystems and Technologies (DEST), 2010 4th IEEE International Conference on*, 400–405. IEEE. ISSN 2150-4938.

[114] Lough, Daniel. "A Taxonomy of Computer Attacks with Applications to Wireless Networks", 2001.

[115] Luke, Sean. "Multiagent Simulation and the MASON Library", August 2011. URL http://cs.gmu.edu/eclab/projects/mason/.

[116] Luke, Sean, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan. "MASON: A New Multi-Agent Simulation Toolkit". *Proceedings of the 2004 Swarmfest Workshop*. 2004.

[117] Lynch, Rajendran K., S. "Design Diagrams for Multi-Agent Systems", 2004.

[118] Lynn III, W.F. "Defending a New Domain: The Pentagon's Cyberstrategy". 2010.

[119] Magoni, D. "nem: A software for network topology analysis and modeling". *10th IEEE International Symposium on Modeling, Analysis, and Simulation of Computer and Telecommunications Systems (MASCOTS)*, 364. 2002.

[120] Manning, ; Raghavan P.;, C and H. Schutze. *Introduction to Information Retrieval*. Cambridge University Press, 2008. ISBN http://nlp.stanford.edu/IR-book/html/htmledition/irbook.html.

[121] McCanne, S., S. Floyd, and K. Fall. "ns2 (network simulator 2)". *last accessed: February*, 23, 2010.

[122] McDonald, C. "The cnet network simulator". *University of Western Australia*, 2003.

[123] McDonald, Chris. "The cnet network simulator". URL http://www.csse.uwa.edu.au/cnet/index.html.

[124] Meadows, Donella H. "Places to Intervene In a System", July 2005.

[125] Medina, A., A. Lakhina, I. Matta, and J. Byers. "BRITE: An approach to universal topology generation". *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, 346–353. IEEE, 2002. ISBN 0769513158.

[126] Meier, J.D. "Improving Web Application Security: Threats and Countermeasures", June 2003. URL http://msdn.microsoft.com/en-us/library/ff649874.aspx.

[127] Miller, D. "DoD works with industry on automated network intrusion defense system", 2011.

[128] Milner, R. and University of Edinburgh. Department of Computer Science. *On relating synchrony and asynchrony*. Department of Computer Science, University of Edinburgh, 1980.

[129] Ming-Yang and Su. "Real-time anomaly detection systems for Denial-of-Service attacks by weighted k-nearest-neighbor classifiers". *Expert Systems with Applications*, 38(4):3492 – 3498, 2011. ISSN 0957-4174. URL http://www.sciencedirect.com/science/article/pii/S0957417410009450.

[130] Mirkovic, J. and P. Reiher. "A taxonomy of DDoS attack and DDoS defense mechanisms". *ACM SIGCOMM Computer Communication Review*, 34(2):39–53, 2004. ISSN 0146-4833.

[131] Mofe, Winnie; Kaeli David; Zhao Qin, Micha; Cheng. "Hunting Trojan Horses", 2006.

[132] Molsa, J. "Mitigating Denial of Service Attacks". *Journal of Computer Security*, volume 13, 807–837. 2005.

[133] Moore, Andrew W., Denis Zuev, and Michael L. Crogan. *Discriminators for use in flow-based classification*. Technical report, Queen Mary University of London, 2005.

[134] Nguyen, Perini A. Bernon C. Pavon J., C. and J. Thangarajah. "Testing in Multi-Agent Systems", July 2009.

[135] Nguyen, Perini A. Tonella P., C.D. "Goal Oriented Testing for MASs", 2008.

[136] Obama, Barack. "The Comprehensive National Cybersecurity Initiative". *White-house Journal*, 2013.

[137] Oetiker, Tobi. "RRDTool". URL oss.oetiker.ch/rrdtool.

[138] Oltean, C., M.; Grosan. "A comparison of several linear genetic programming techniques", 2003.

[139] Park, Hyungwook and Paul A. Fishwick. "A GPU-Based Application Framework Supporting Fast Discrete-Event Simulation". *Simulation*, 86:613–628, October 2010. ISSN 0037-5497. URL http://dx.doi.org/10.1177/0037549709340781.

[140] Parker, S.P. *McGraw-Hill dictionary of scientific and technical terms*. MCGRAW HILL DICTIONARY OF SCIENTIFIC AND TECHNICAL TERMS. McGraw-Hill, 2003. ISBN 9780070423138. URL http://books.google.com/books?id=xOPzO5HVFfEC.

[141] Peng, Tao. *Defending Against Distributed Denial of Service Attacks*. Ph.D. thesis, The University of Melbourne, April 2004.

[142] Perdisci, Roberto, Davide Ariu, Prahlad Fogla, Giorgio Giacinto, and Wenke Lee. "McPAD : A Multiple Classifier System for Accurate Payload-based Anomaly Detection". *Computer Networks, Special Issue on Traffic Classification and Its Applications to Modern Networks*, 5:864–881, 2009.

[143] Postel, J. "RFC 793: Transmission control protocol", 1981.

[144] Postel, J.B. "User Datagram Protocol. RFC 768", 1980.

[145] Poutakidis, Winikoff M. Padgham L. Zhang Z., D. "Debigging and Testing of Multi-Agent Systems using Design Artefacts", 2009.

[146] Quittek, J., T. Zseby, and B. Claise. *S. Zander," Requirements for IP Flow Information Export (IPFIX)*. Technical report, RFC 3917, October 2004.

[147] Resnick, P. and R. Zeckhauser. "Trust among strangers in Internet transactions: Empirical analysis of eBay's reputation system". *Advances in Applied Microeconomics: A Research Annual*, 11:127–157, 2002. ISSN 0278-0984.

[148] Rish, Irina. "An empirical study of the naive Bayes classifier", 2001.

[149] Russell, Stuart and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2nd edition, December 2002. ISBN 0137903952.

[150] Scepanovic, Sanja. *Mitigating DDoS attacks with cluster-based filtering*. Master's thesis, Aalto University, June 2011.

[151] Sellke, S.H., N.B. Shroff, and S. Bagchi. "Modeling and automated containment of worms". *IEEE Transactions on Dependable and Secure Computing*, 71–86, 2007. ISSN 1545-5971.

[152] Serrano, Gomez-Sanz J.J. Botia J.A. Pavon J., E. "Intelligent Data Analysis, Applied to Debug Complex Software Systems", 2009.

[153] Shawe-Taylor, John and Nello Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[154] Shirey, R. "Internet Security Glossary, Version 2", August 2007. URL http://tools.ietf.org/html/rfc4949.

[155] Shoham, Y. and K. Leyton-Brown. *Multiagent systems: algorithmic, game-theoretic, and logical foundations*. Cambridge Univ Pr, 2008. ISBN 0521899435.

[156] Sierra, Aguilar-J.A.R. Norieaga P. Esteva M. Arcos J.L., C. "Engineering Multi-Agent Systems as Electronic Institutions", 2004.

[157] Singh, Ram Kumar. "Intrusion Detection System Using Advanced Honeypots", 2009.

[158] Skormin, O.; Tokhtabayev A.;, V.; Shiryayeva and J. Moronski. "Towards Fully Automatic Defense Mechanism for a Computer Network Emulating Active Immune Response", 2006.

[159] Skoudis, Ed and Tom Liston. *Counter Hack Reloaded*. Prentice Hall, 2nd edition, January 2006.

[160] Soergel, D. "jlibsvm", 2009. URL http://dev.davidsoergel.com/trac/jlibsvm.

[161] Spatharis, A., I. Foudalis, M. Gjoka, P. Krouska, C. Amanatidis, C. Papadimitriou, and M. Sideri. "Improved tradeoff-based models of the Internet". *Proc. of SIWN/IEEE International Conference on Complex Open Distributed Systems*, volume 7. 2008.

[162] Specht, Ruby B., Stephen M.and Lee. "Distributed Denial of Service: Taxonomies of Attacks, Tools and Countermeasures". *Proceedings of the 17th International Conference on Parallel and Distributed Computing Systems, 2004 International Workshop on Security in Parallel and Distributed Systems*, 543–550. September 2004.

[163] Sperotto, A., G. Schaffrath, R. Sadre, C. Morariu, A. Pras, and B. Stiller. "An Overview of IP Flow-Based Intrusion Detection". *IEEE Communications Surveys & Tutorials*, 12(3):343–356, 2010. ISSN 1553-877X.

[164] Stanley, Kenneth O. "Real-Time Neuroevolution in the NERO Video Game", 2005.

[165] Storn, Rainer and Kenneth Price. "Differential Evolution A Simple and Efficient Heuristic for global Optimization over Continuous Spaces". *Journal of Global Optimization*, 11:341–359, 1997. ISSN 0925-5001. URL http://dx.doi.org/10.1023/A%3A1008202821328.

[166] von Stosch, Moritz. "Novel Strategies for Process Control Based on Hybrid Semi-parametric Mathematical Systems", 2011.

[167] Subrahmanian, V.S. *Heterogeneous Agent Systems*. Mit Press, 2000. ISBN 9780262194365. URL http://books.google.com/books?id=2YMUNKuq8mYC.

[168] Talbi, E.G. *Metaheuristics: from design to implementation*. Wiley, 2009. ISBN 0470278587.

[169] Tartakovsky, B. L.; Blazek R.B., A. G.; Rozovskii. "A Novel Approach to Detection of Intrusions in Computer Networks via Adaptive Sequential and Batch-Sequential Change-Point Detection Methods", 2006.

[170] Thomas, Ciza. "Usefulness of DARPA Dataset for Intrusion Detection System Evaluation". *IEEE*. Indian Institute of Science, Bangalore, India, 2011.

[171] Urbanowicz, Ryan J. and Jason H. Moore. "Learning Classifier Systems: A Complete Introduction, Review, and Roadmap", 2009.

[172] Vapnik, V.N. *The nature of statistical learning theory*. Springer Verlag, 2000. ISBN 0387987800.

[173] Vapnik, V.N. "An overview of statistical learning theory". *Neural Networks, IEEE Transactions on*, 10(5):988–999, 2002. ISSN 1045-9227.

[174] Varga, A. et al. "The OMNeT++ discrete event simulation system". *Proceedings of the European Simulation Multiconference (ESM2001)*, 319–324. 2001.

[175] Viola, Ivan, Armin Kanitsar, and Meister Eduard Gröller. "Importance-Driven Feature Enhancement in Volume Visualization". *IEEE Transactions on Visualization and Computer Graphics*, 11(4):408–418, 2005. URL http://www.cg.tuwien.ac.at/research/publications/2005/viola-2005-imp/.

[176] Wagner, A., T. D
"ubendorfer, B. Plattner, and R. Hiestand. "Experiences with worm propagation simulations". *Proceedings of the 2003 ACM workshop on Rapid Malcode*, 34–41. ACM, 2003. ISBN 1581137850.

[177] Wang, D., G. Chang, X. Feng, and Guo R. "Research on the Detection of Distributed Denial of Service Attacks Based on the Characteristics of IP Flow". *NPC Proceedings of the IFIP International Conference on Network and Parallel Computing*, 1:86–93, 2008.

[178] Wang, Ke, Gabriela Cretu, and Salvatore Stolfo. *Anomalous Payload-based Worm Detection and Signature Generation*. Technical report, Columbia University, 2004.

[179] Wang, Shen Y. Huang T. Zeng Z., H. *The sixth International Symposium on Neural Networks (ISNN 2009*. Springer, Berlin Heidelberg, 2009. ISBN 9783642012150.

[180] Waxman, BM. "Routing of Multipoint Connections". *Selected Areas in Comm*, 6(9):1617–1622, 1988.

[181] Weiss. *Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence*.

[182] Whitman, Michael E. and Herbert J. Mattord. *Principles of Information Security*. Course Technology, 2011.

[183] Willinger, W., D. Alderson, and J. Doyle. "Mathematics and the Internet: a source of enormous confusion and great potential". *Notices of the American Mathematical Society*, 56(5):586–599, May 2009.

[184] Willinger, W. and V. Paxson. "Where mathematics meets the Internet". *Notices of the American Mathematical Society*, 45(8):961–971, 1998. ISSN 0002-9920.

[185] Wilson, Timothy. *MFIRE-2: A Multi-Agent System for Flow-Based Intrusion Detection Using Stochastic Search*. Master's thesis, Air Force Institute of Technology, March 2012.

[186] Winick, J. and S. Jamin. *Inet-3.0: Internet Topology Generator*. Technical Report UM-CSE-TR-456-02, Department of EECS, University of Michigan, 2002.

[187] Winter, P., E. Hermann, and M. Zeilinger. "Inductive Intrusion Detection in Flow-Based Network Data Using One-Class Support Vector Machines". *Proc. 4th IFIP Int New Technologies, Mobility and Security (NTMS) Conf*, 1–5. 2011.

[188] Yook, S.H., H. Jeong, and A.L. Barabási. "Modeling the Internet's large-scale topology". *Proceedings of the National Academy of Sciences of the United States of America*, 99(21):13382, 2002.

[189] Zacharia, G. and P. Maes. "Trust management through reputation mechanisms". *Applied Artificial Intelligence*, 14(9):881–907, 2000. ISSN 0883-9514.

[190] Zegura, E.W., K.L. Calvert, and M.J. Donahoo. "A quantitative comparison of graph-based models for Internet topology". *IEEE/ACM Transactions on Networking (TON)*, 5(6):770–783, 1997. ISSN 1063-6692.

[191] Zeng, X., R. Bagrodia, and M. Gerla. "GloMoSim: a library for parallel simulation of large-scale wireless networks". *ACM SIGSIM Simulation Digest*, 28(1):154–161, 1998. ISSN 0163-6103.

[192] Zhang, Ruishan, Xinyuan Wang, Ryan Farley, Xiaohui Yang, and Xuxian Jiang. "On the feasibility of launching the man-in-the-middle attacks on VoIP from remote attackers". *Proceedings of the 4th International Symposium on Information, Computer, and Communications Security*, ASIACCS '09, 61–69. ACM, New York, NY, USA, 2009. ISBN 978-1-60558-394-5. URL http://doi.acm.org/10.1145/1533057.1533069.

[193] Zou, C.C., W. Gong, and D. Towsley. "Code red worm propagation modeling and analysis". *Proceedings of the 9th ACM conference on Computer and communications security*, 138–147. ACM, 2002. ISBN 1581136129.

[194] Zou, C.C., W. Gong, and D. Towsley. "Worm propagation modeling and analysis under dynamic quarantine defense". *Proceedings of the 2003 ACM workshop on Rapid Malcode*, 60. ACM, 2003. ISBN 1581137850.

Vita

David Ryan was born February 22, 1989 in Cincinnati, Ohio. When David was 18 he attended the United States Air Force Academy and Graduated in 2011 with a Bachelor of Science in Computer Engineering. After graduation from the Academy, David attended the Air Force Institute of Technology at Wright-Patterson Air Force Base, earning a Master of Science in Computer Engineering with an emphasis in Artificial Intelligence. David begins his first operational assignment with the 463rd Network Warfare Squadron at Lackland Air Force Base as an Electronic Warfare Officer following Graduation.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704–0188

1. REPORT DATE *(DD–MM–YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From — To)*
21–03–2013	Master's Thesis	Oct 2011–Mar 2013

4. TITLE AND SUBTITLE

A Multi Agent System for Flow-Based Intrusion Detection

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)

Ryan, David A., Second Lieutenant, USAF

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)

Air Force Institute of Technology
Graduate School of Engineering and Management (AFIT/EN)
2950 Hobson Way
WPAFB, OH 45433-7765

8. PERFORMING ORGANIZATION REPORT NUMBER

AFIT-ENG-13-M-43

9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)

Intentionally Left Blank

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION / AVAILABILITY STATEMENT

DISTRIBUTION STATEMENT A.
APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The detection and elimination of threats to cyber security is essential for system functionality, protection of valuable information, and preventing costly destruction of assets. This thesis presents a Mobile Multi-Agent Flow-Based IDS called MFIREv3 that provides network *anomaly detection* of intrusions and automated defense.

This version of the MFIRE system includes the development and testing of a Multi-Objective Evolutionary Algorithm (MOEA) for feature selection that provides agents with the "optimal" set of features for classifying the state of the network. Feature selection provides separable data points for the selected attacks: Worm, Distributed Denial of Service, Man-in-the-Middle, Scan, and Trojan.

This investigation develops three techniques of self-organization for multiple distributed agents in an intrusion detection system: Reputation, Stochastic, and Maximum Cover. These three movement models are tested for effectiveness in locating good agent vantage points within the network to classify the state of the network.

MFIREv3 also introduces the design of defensive measures to limit the effects of network attacks. Defensive measures included in this research are rate-limiting and elimination of infected nodes.

The results of this research provide an optimistic outlook for flow-based multi-agent systems for cyber security. The impact of this research illustrates how feature selection in cooperation with movement models for multi agent systems provides excellent attack detection and classification.

15. SUBJECT TERMS

Autonomous Multi-Agent Network Intrusion Detection

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			(ENG) Dr. Gary B. Lamont, AFIT/ENG
U	U	U	UU	213	19b. TELEPHONE NUMBER *(include area code)* (937) 255-3636 ext. 9363